PRICING BY LOCAL SEARCH IN COLUMN GENERATION FOR THE AIRLINE CREW PAIRING PROBLEM

by

NİMET AKSOY

Submitted to the Graduate School of Engineering and Natural Sciences
in partial fulfillment of
the requirements for the degree of
Master of Science

Sabancı University
August 2010

PRICING BY LOCAL SEARCH IN COLUMN GENERATION
FOR THE AIRLINE CREW PAIRING PROBLEM

APPROVED BY

Assoc. Prof. Ş. İlker Birbil (Thesis Advisor)	.....
Assist. Prof. Kerem Bülbül (Thesis Co-advisor)	.....
Assist. Prof. Güvenç Şahin	.....
Assoc. Prof. Tonguç Ünlüyurt	.....
Assist. Prof. Hüsnü Yenigün	.....

DATE OF APPROVAL: 09.08.2010

to my beloved family

Acknowledgments

First of all, I would like to express my deepest gratitude to my thesis advisors, whose expertise, understanding and guidance added a lot to my graduate experience. I am very grateful to Assoc. Prof. Ş. İlker Birbil for his confidence in me and his encouraging supervision, which relieves me even under most stressful conditions. I owe my sincere thanks to Assist. Prof. Kerem Bülbül for his continuous understanding and endless support throughout my undergraduate and graduate studies at Sabancı University. I am indebted to Assist. Prof. Hüsnü Yenigün for his invaluable contribution to my thesis and his helpfulness. I am also thankful to Assist. Prof. Güvenç Şahin and Assoc. Prof. Tonguç Ünlüyurt for being actively interested in my studies since my undergraduate years and their willingness to give helpful advice whenever I needed.

I would like to thank TÜBİTAK BİDEB for providing me financial support during my graduate studies.

I must acknowledge the assistance of my colleagues Erdal Mutlu and İbrahim Muter. Their experience in coding and the airline crew pairing problem, and their friendly attitude made the implementation phase a lot easier for me. Another thing that made my life much easier was the cordial and joyful office atmosphere, and very special thanks go to my colleagues at FENS1021 and FENS1028. I am especially thankful to Özlem Çoban, Gizem Kılıçaslan, Elif Özdemir, Merve Şeker, Semih Yalçındağ and Mahir Yıldırım. They have been much more than office mates to me, and I enjoyed every moment of two years we have shared.

Words are not enough to express my thankfulness to my dearest friends Nazlı Aktakke, Gizem Bayramoğlu, Merve Gümüş and Ece Şuşut. Their presence has given me happiness and strength since my childhood, and I wish to feel their love and support forever.

And finally, I doubt that I will ever be able to convey my appreciation fully, but I must mention my eternal gratitude to my beloved family. Without them supporting and trusting me by all means, and their neverending love, not only this thesis, but I myself would not be where I am, at all.

PRICING BY LOCAL SEARCH IN COLUMN GENERATION FOR THE AIRLINE CREW PAIRING PROBLEM

Nimet Aksoy

Industrial Engineering, Master of Science Thesis, 2010

Thesis Advisors: Assoc. Prof. Ş. İlker Birbil, Assist. Prof. Kerem Bülbül

Keywords: crew pairing, column generation, pricing subproblem, multi-label shortest path problem, flight network, duty network, local search

Abstract

The airline crew pairing problem (ACP) is finding the least costly set of crew pairings so that each flight given in the flight schedule is covered. The ACP is traditionally modeled either as a set partitioning problem or a set covering problem. Due to the large number of possible pairings (columns), these models are usually solved by the column generation (CG) method. For the ACP, the pricing subproblem of the CG corresponds to a multi-label shortest path problem (MLSP) typically solved over a flight network. The MLSP over the flight network is *NP*-hard and it suffers from an exponential complexity even for moderate size flight networks.

In order to overcome the complexity of the pricing subproblem, we propose a column generation method to solve the ACP, in which a hybrid pricing procedure is used. In this hybrid procedure, the pricing subproblem consists of three steps. First, we apply a local search mechanism on the partial duty period pool to construct pairings with negative reduced cost. In cases local search mechanism cannot find such a pairing, we execute a heuristic MLSP algorithm over the partial duty network to price out negative reduced cost pairings for the restricted master problem (RMP). If this method also fails, we solve the MLSP over the flight network. By adopting this hybrid approach, we aim to decrease the number of CG iterations where the MLSP is executed over the flight network, and reduce the computation time per iteration as well as the total computation time. We test the efficiency of our approach on real-life instances acquired from a local airline company, and present numerical results.

HAVAYOLU EKİP EŞLEME PROBLEMİNDE KOLON TÜRETME YÖNTEMİNİN YEREL ARAMA İLE ÜCRETLENDİRİLMESİ

Nimet Aksoy

Endüstri Mühendisliği, Yüksek Lisans Tezi, 2010

Tez Danışmanları: Doç. Dr. Ş. İlker Birbil, Yrd. Doç. Dr. Kerem Bülbül

Anahtar Kelimeler: ekip eşleme, kolon türetme, ücretlendirme altproblemi, çok takılı en kısa yol problemi, uçuş serimi, uçuş görev serimi, yerel arama

Özet

Havayolu ekip eşleme problemi uçuş çizelgesindeki her bir uçuşun kapsanmasını sağlayacak şekilde en az maliyetli ekip eşlemelerinin bulunmasıdır. Bu problem literatürde genel olarak küme bölüntüleme veya küme kapsama problemi olarak modellenmektedir. Olası eşlemelerin (kolonların) sayısındaki fazlalık nedeniyle bu modellerin kolon türetme yöntemiyle çözülmesi sıkça başvurulan bir yaklaşımdır. Havayolu ekip eşleme problemi için kolon türetme yönteminin ücretlendirme altproblemi uçuş serimi üzerinde çözülen çok takılı en kısa yol problemine karşılık gelmektedir. Uçuş serimi üzerinde çözülen çok takılı en kısa yol problemi NP-zor bir problemdir ve karmaşıklığı orta büyüklükteki çizelgeler için bile üsseldir.

Bu çalışmada, havayolu ekip eşleme problemini, ücretlendirme altprobleminin karmaşıklığını azaltmak amacıyla melez bir ücretlendirme prosedürü kullanarak, kolon türetme yöntemiyle çözmekteyiz. Önerdiğimiz melez prosedürde, ücretlendirme altproblemi üç adımdan oluşmaktadır. İlk olarak, kısmi uçuş görev havuzu üzerinde uyguladığımız yerel arama yöntemi ile negatif azaltılmış maliyetli eşlemeler oluşturmaya çalışmaktayız. Yerel arama yönteminin böyle bir eşleme oluşturamadığı durumlarda çok takılı en kısa yol problemini kısmi uçuş görev serimi üzerinde çözmekte ve kısıtlı ana probleme eklemek üzere negatif azaltılmış maliyetli eşlemeler aramaktayız. Bu metodun da kısıtlı ana probleme eklenecek eşleme bulamadığı durumlarda çok takılı en kısa yol problemini uçuş serimi üzerinde çözmekteyiz. Benimsediğimiz melez yaklaşım sayesinde, çok takılı en kısa yol probleminin uçuş serimi üzerinde çözüldüğü kolon türetme iterasyonlarının sayısını ve iterasyon başına düşen çözüm süresi ile toplam çözüm süresini azaltmayı amaçlamaktayız. Yaklaşımımızın etkinliğini yerel bir havayolu şirketinden edindiğimiz örnek problemler üzerinde test etmekte ve sayısal sonuçlar sunmaktayız.

Table of Contents

Abstract **vi**

Özet **vii**

1 INTRODUCTION **1**
1.1 Contributions of This Study 2
1.2 Outline 4

2 LITERATURE REVIEW **5**
2.1 Terms and Definitions 5
2.2 Solving the LP Relaxation 7
2.3 The Pricing Subproblem 9

3 PRICING BY LOCAL SEARCH IN COLUMN GENERATION FOR THE AIRLINE CREW PAIRING PROBLEM **13**
3.1 Problem Statement 13
3.2 Initial Feasible Pairing Pool Generation 17
3.3 The Pricing Subproblem 18
3.3.1 Pricing by Multi-Label Shortest Path Algorithm on the Flight Network 20
3.3.2 Pricing by Multi-Label Shortest Path Algorithm on the Duty Network 23
3.3.3 Pricing by A Local Search Algorithm 25
3.4 Implementation Details 27
3.4.1 Class Design and Structures 27
3.4.2 Generic Programming with Boost Libraries 28

4 COMPUTATIONAL RESULTS **30**

5 CONCLUSION **42**

Bibliography **43**

A Class Diagrams and Structures **46**

List of Figures

2.1 A sequence of flight legs in a duty period (DP). 5
2.2 A sequence of duty periods in a pairing. 6

3.1 A small flight network example. 14
3.2 A small duty network example. 14
3.3 Flow chart of the proposed algorithm. 16
3.4 Label structure for the MLSP. 21
3.5 Local search method. 26

4.1 CPU times (sec.) per iteration for DS1. 34
4.2 CPU times (sec.) per iteration for DS2. 34
4.3 CPU times (sec.) per iteration for DS3. 35
4.4 CPU times (sec.) per iteration for DS4. 35
4.5 Comparison of HYBRID and PURE by changes in the objective function value for DS1. 39
4.6 Comparison of HYBRID and PURE by changes in the objective function value for DS2. 40
4.7 Comparison of HYBRID and PURE by changes in the objective function value for DS3. 40
4.8 Comparison of HYBRID and PURE by changes in the objective function value for DS4. 41

A.1 *Duty* and *Flight* as *SeqOfFlightlegs*. 46
A.2 *Connection*, *Flightleg* and *Pairing* classes. 46
A.3 Class diagram for the *Label* structure. 47
A.4 *NodeLabel* and *PathLabel* classes. 48

List of Tables

3.1 Minimum rest time between two duty periods. 15
3.2 Maximum elapsed time for a duty period. 15

4.1 CPU times needed to generate an initial feasible solution. 32
4.2 Initial and (LP and IP) optimal objective function values. 32
4.3 Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS1. 36
4.4 Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS2. 36
4.5 Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS3. 36
4.6 Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS4. 37
4.7 Some performance measures after turning off the LS mechanism. . . . 37
4.8 Some performance measures for the proposed pricing method with the original initial feasible pairing pool technique and the modified version (with less number of duty periods). 38
4.9 Some performance measures acquired by turning off the LS and DN-MLSP mechanisms after the iteration at which FNMLSP is first executed. 39

CHAPTER 1

INTRODUCTION

The airline crew scheduling problem consists of assigning crew members to a given timetable of flights, such that the assignment is feasible with respect to civil aviation rules, and the crew costs are minimized. Crew scheduling is both challenging and important for an airline company for two reasons. First, following fuel costs, crew costs are the second largest direct operating cost of airlines [3]. Thus, crew scheduling optimization has the potential to provide significant cost reduction for airline companies. Second, the strict regulations imposed by the government and the large scale of the problem makes optimal airline crew scheduling a complex task to accomplish. While trying to preserve feasibility, time efficiency of the solution should also be considered. The crew scheduling problem consists of two subproblems: the crew pairing problem and the crew assignment problem. As the first phase, the crew pairing problem is solved in order to find the least costly set of a sequence of flights (i.e. pairings) starting and ending at the same airport (i.e. crew base) and covering all flights given in the timetable. Pairings generated by the solution of the crew pairing problem are given as input to the crew assignment problem. The crew members are optimally allocated to these pairings in a way that a set of feasibility constraints are satisfied and employee satisfaction is maximized. In this thesis, the crew pairing problem is considered.

The crew pairing problem is traditionally modeled either as a set partitioning problem or a set covering problem. The integer programming (IP) formulation for the set covering problem is as follows:

$$\text{minimize} \quad \sum_{j \in \mathcal{P}} c_j x_j \tag{1.1}$$

$$\text{subject to} \quad \sum_{j \in \mathcal{P}} a_{ij} x_j \geq 1, \quad i \in \mathcal{F}, \tag{1.2}$$

$$x_j \in \{0,1\}, \quad j \in \mathcal{P}, \tag{1.3}$$

where $\mathcal{P}$ is the set of all feasible pairings, $\mathcal{F}$ is the set of all flights, c_j is the cost of

pairing j, $a_{ij} = 1$ if flight i is covered by pairing j and 0; otherwise. If pairing j is selected by the solution, its corresponding variable $x_j = 1$, and otherwise it is set to 0. The only difference between the set partitioning and the set covering formulations is that the inequality in (1.2) is changed to equality in the set partitioning model. This means that the flights are covered exactly once by the set partitioning formulation, whereas they are allowed to be covered more than once by the set covering formulation. In other words, the set covering model allows for deadhead flights. That is, one crew is assigned to cover the flight, the other crews fly as passengers in order to reach their destination airport to continue with their own pairings.

This study proposes a solution method to the crew paring problem based on the LP relaxation of the set covering model. Due to the nature of the crew pairing problem, the search space might become intractably large for even a moderate number of flights and generating all feasible pairings might be very costly in terms of computation time. Furthermore, as more pairings are generated, solving the LP relaxation becomes harder. Therefore, usually, a column generation approach is adopted for the solution of the crew pairing problem. In this approach, at each iteration, a restricted master problem (RMP), which includes only a subset of all possible pairings, is solved. After solving the RMP, the pricing procedure is applied in order to find a pairing with negative reduced cost. If such a pairing is found, it is added to the problem and the RMP is resolved. Same steps are continued until no negative reduced cost pairing is found, i.e., the optimal solution is reached. The pricing subproblem depends on the problem characteristics. In the crew pairing problem, the pricing subproblem corresponds to a multi-label shortest path (MLSP) problem, which is typically solved over a flight network. Several labels related with the feasibility and the cost of the pairings are kept throughout the paths on the network, and once a negative reduced cost path is found, it is added as a new column to the RMP. This procedure is repeated until MLSP fails to find a negative reduced cost column.

1.1 Contributions of This Study

In this thesis, we focus on column generation to solve the crew pairing problem. While solving the crew pairing problem by column generation, modeling the pricing subproblem as a multi-label shortest path problem (MLSP) is common practice. However, the MLSP subproblem is usually the most time-consuming step of the column generation approach and its complexity is exponential in the number of flights in the schedule.

Thus, dealing with large-scale networks is difficult. This study proposes an alternating pricing procedure to overcome this difficulty. We are motivated by the existence of a diverse and rich duty period pool generated during the pairing generation phase, and we use this duty period pool to apply a local search based mechanism to price out the negative reduced cost pairings. The pool is populated with duty periods during the initial feasible solution construction (See Section 3.2) and updated each time a new pairing is generated at each iteration. Since this mechanism yields a heuristic method, it is not guaranteed that the local search finds a pairing that will improve the RMP objective function value at each iteration even if such a pairing exists. In that case, we resort to MLSP over the partial duty network that is constructed from the populated duty period pool. In the duty network, the number of feasibility rules that need to be tracked are reduced since the feasibility rules related to a duty period (maximum elapsed time, minimum/maximum rest time etc.) are already satisfied during the duty network construction. Since it is impossible to generate all possible duty periods at once, each MLSP iteration is executed on the partial duty network. Moreover, as the number of feasibility rules is reduced, the number of necessary labels is also reduced. Less number of labels and the partiality of the duty network provide an easier solution to the MLSP problem, compared to solving it over the entire flight network. However, MLSP solved over the duty network might also fail to find a negative reduced cost pairing. In such a case, MLSP is executed over the flight network. This last alternative is the most exhaustive way to price out new pairings; but, it is required to ensure the optimality of the subproblem. Our main aim is to minimize the number of costly calls to the MLSP solution over the flight network. In the ideal case, we wish to solve the costly problem only once, at the last iteration, to ensure optimality.

Contributions of this study can be listed as follows:

- Proposed local search procedure is a simple and easy-to-implement heuristic which rapidly provides negative reduced cost pairings for the RMP.
- Pricing with local search and utilizing the partial duty network are time-efficient alternatives to pricing with MLSP over the flight network in terms of time spent per subproblem iteration.
- We ensure the optimality of the subproblem by resorting to MLSP over the flight network which is an exact method to find negative reduced cost pairings whenever the local search algorithm and MLSP over the duty network fail to do so.

- We test our approach on different real-life problems and present numerical results. We benchmark our hybrid scheme against a pure MLSP pricing scheme. Based on our comparisons, we observe the following:
  - Incorporating local search and the duty network reduces the number of calls to exact MLSP solved over the flight network.
  - Total CPU times are reduced.
- We adopt a generic implementation scheme using Boost C++ Graph and Date-Time libraries [7]. Our class structures are compatible with these libraries which makes it easier to work with network related problems.

1.2 Outline

The outline of this thesis is as follows: **Chapter 2** gives a literature review on the crew pairing problem, the column generation method and the pricing subproblem. **Chapter 3** includes our problem statement and proposed pricing scheme which alternates between MLSP and local search as well as the implementation details of our method. In **Chapter 4**, we present a computational study on a set of actual data acquired from a local airline company. Conclusions are discussed in **Chapter 5**.

CHAPTER 2

LITERATURE REVIEW

2.1 Terms and Definitions

Throughout this study, we frequently make use of some standard terms used in airline scheduling problems. Before reviewing the crew pairing problem literature, we need to define some important terminology. We refer to Vance *et al.* [26] and Barnhart *et al.* [5] for the following definitions:

Flight Leg (Segment): A single nonstop flight. Two flight legs could be connected if and only if the arrival airport of the first leg is the same with the departure airport of the second leg, and the time between these flights respects the predefined feasibility rules.

Duty Period: A sequence of flight legs with short time periods separating them. A duty period might also be defined as a single work day for a crew. Duty periods are also connected to each other according to some regulations. Figure 2.1 illustrates the structure of a duty period.

Crew Base (Domicile): The city where the crews are stationed.

Pairing: A sequence of duty periods with overnight rests between them. Each

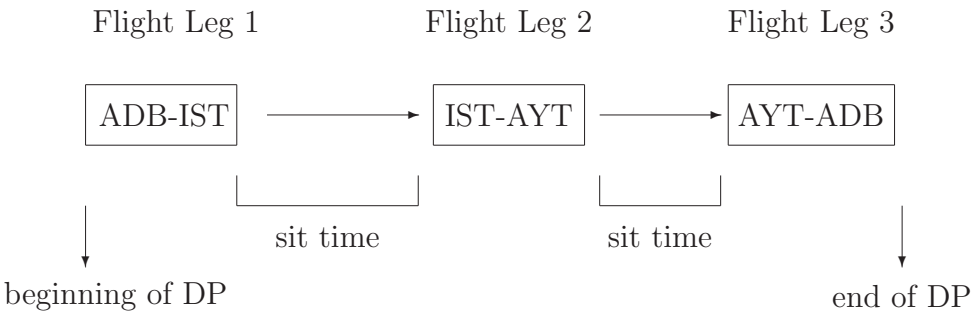

Figure 2.1: A sequence of flight legs in a duty period (DP).

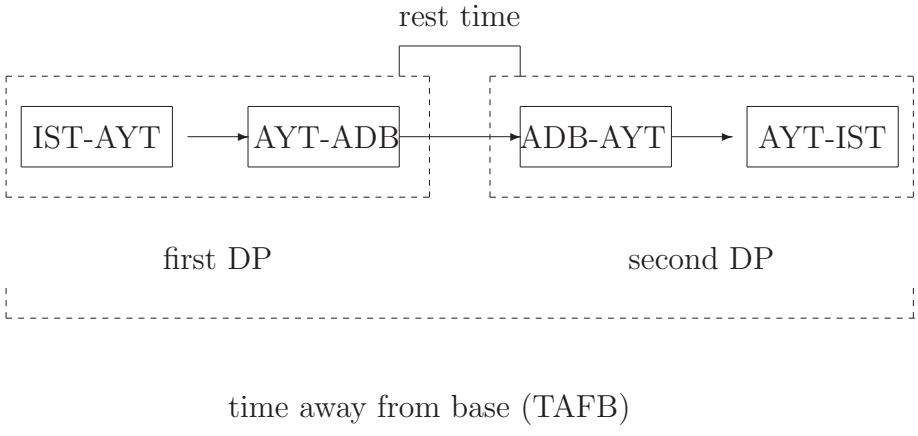

Figure 2.2: A sequence of duty periods in a pairing.

pairing must begin and end at the same crew base. Figure 2.2 exhibits the structures and terminology related to a pairing.

Sit Time: The time between two flight legs within a duty period.

Rest Time (Layover): The time between two duty periods within a pairing.

In the crew pairing problem, flight network construction is strictly dependent on a number of constraints. Kornilakis and Stamatopoulos [20] identify these constraints as:

1. **Temporal Constraints:** The departure of a flight leg/ duty period should take place after the arrival of the previous flight leg/duty period.

2. **Spatial Constraints:** For every two consecutive flight legs/duty periods within a pairing, the second must depart from the airport that the first arrives at. For a pairing, the departure city of its first leg and the arrival city of its last leg must be the crew base.

3. **Fleet Constraints:** Cockpit crew is usually designated to operate only one type of aircraft whereas the cabin crew may be assigned to several types of aircraft. Therefore, all flights in a pairing generated for cockpit crew must be flown by the same type of aircraft. As a consequence, pairings that are legal for the cabin crew may be illegal for the cockpit crew and this makes the cockpit crew problem easier to solve.

4. **Constraints due to Laws and Regulations:** In order to remain legal, constructed pairings must respect the regulations specified by government laws and collective agreements. These regulations are usually related to the maximum duration of a duty period, the maximum flight time allowed over a time period, the

minimum length of the rest time between two duties and so on.

The last set of constraints might vary in different types of problem settings. However, the basic feasibility rules associated with duty periods and pairings might be stated as follows:

- The time between two flights in a duty period should be greater than or equal to the *minimum sit time* and less than or equal to the *maximum sit time*.
- The total elapsed time of a duty period should be less than or equal to the *maximum elapsed time* defined for the duty and the total flying time of a duty period is restricted by the *maximum flying time*.
- The number of duty periods in a pairing cannot be greater than the *maximum number of duty periods* defined for the pairing.
- The time between two duty periods in a pairing should be greater than or equal to the *minimum rest time* and less than or equal to the *maximum rest time*.
- *Maximum time away from base* is an upper limit on the total elapsed time of a pairing.

2.2 Solving the LP Relaxation

In the literature, solution approaches for the crew pairing problem are generally based on solving the LP relaxation of the set covering or the set partitioning models. Anbil *et al.* [1] addresses the crew pairing problem by introducing the *Trip Reevaluation and Improvement Program (TRIP)* method for iterative pairing generation and subproblem optimization. This method starts with an initial solution (usually generated by modifying the previous period's solution) and continues with selecting a set of pairings and solving a subproblem at each iteration. If a set of pairings that yield a better solution is found, they replace the chosen set of old pairings. The drawback of the TRIP is that since it is a subproblem optimization approach, it may stall at a suboptimal solution that cannot be improved further. However, another method they utilize in conjunction with TRIP is a global approach called *SPRINT* which is able to find the optimal solution for the LP relaxation. SPRINT begins with a subproblem consisting of a subset of columns. The dual values acquired after solving the subproblem are used to calculate the reduced cost for pricing out new columns to the subproblem. This procedure

is carried out until no negative reduced cost columns can be found, i.e. the optimal LP solution is found. SPRINT method is proved to solve a problem with 5.5 million columns by solving only 25 subproblems. Bixby *et al.* [6] deals with very large-scale linear programs by proposing a combination of interior point and simplex methods. Similar to the column generation approach, this method is based on a process called *sifting*, which is originally suggested by Forrest [16]. It should be noted that, SPRINT is the practical term used for sifting.

Column generation is a widely used method to solve the LP relaxation of the crew pairing problem. The set covering model of the crew pairing problem corresponds to the *master problem* of the column generation since it contains all of the possible pairings. Due to the intractably large number of variables (pairings) in the master problem, column generation method works on a feasible subset of all pairings and the problem with this subset of columns is called the *restricted master problem (RMP)*. Main steps for the column generation procedure are given as follows:

Step 1: The restricted master problem is solved to find an optimal solution for the current set of columns.

Step 2: The *pricing subproblem* is solved to find a column that may improve the incumbent solution. This corresponds to finding a column with a negative reduced cost.

Step 3: If an improving column can be found at **Step 2**, it is added to the RMP and the procedure restarts from **Step 1**. If **Step 2** is not successful, the algorithm is stopped since the optimal solution for the LP relaxation is found.

Crainic and Rousseau [9] solve the set covering formulation of the airline crew pairing problem using a column generation technique. Their algorithm starts with a set of one-day pairings and solves the subproblem over these pairings to find a pairing that improves the solution. At each iteration, they increase the number of duty periods in a pairing by one and check whether a negative reduced cost pairing is found. The column generation iterations continue until no negative reduced cost pairing is found. Anbil *et al.* [2] adopt a column generation approach towards the airline crew pairing problem which combines SPRINT with column generation.

Besides the crew pairing problem, column generation is utilized in order to solve other types of crew scheduling problems. Desrochers and Soumis [13] use the column generation to solve the urban transit crew scheduling problem whereas Desaulniers *et*

al. [10] deal with crew scheduling problems, and Gamache *et al.* [17] focus on the crew rostering problem.

Branch-and-price (B&P) method, which is a variant of the *branch-and-bound (B&B)* technique, is another solution approach for the problems with a large variable set. In B&P, column generation is applied at each node of the B&B tree. If negative reduced cost columns are found after pricing, they are added to the RMP and RMP is reoptimized. If such columns cannot be found and the incumbent solution is not integer, a branching procedure is applied. For the airline crew pairing problem, B&P is adopted by Vance *et al.* [26]. They approximately solve the pricing subproblem of the column generation and thus, acquire near-optimal solutions. Savelsbergh and Sol [23] propose a B&P approach for the general pickup and delivery problem. We refer to Barnhart *et al.* [4] for an extensive study on B&P method applied to several types of problems along with different formulations.

2.3 The Pricing Subproblem

For the column generation solution of the LP relaxation of the crew pairing problem, the pricing subproblem usually corresponds to the *multi-label shortest path problem (MLSP)* or the *constrained shortest path problem (CSP)*. Labels and constraints are essential for the feasibility tracking of the paths and the shortest paths are found to determine the negative reduced cost pairings. The pricing subproblem is solved at each iteration of the column generation in order to find a pairing that would improve the RMP objective function unless there is degeneracy.

In the literature, the constrained shortest path problem is frequently utilized in order to solve vehicle routing problems with constraints. Dynamic programming is used by Desrochers and Soumis [12] for the *shortest path problem with time windows (SPPTW)*. Their algorithm is constructed by adjusting the Ford-Bellman-Moore dynamic programming algorithm, which is originally designed for the regular shortest path problem. They try to find the minimum-cost path from the source node to the sink respecting the time windows specified for each node. Desrochers *et al.* [14] formulate the *vehicle routing problem with time windows (VRPTW)* as a set partitioning model and apply column generation to the LP relaxation of this formulation. Unlike the regular vehicle routing problem, the vehicle routing problem with time windows requires keeping track of the allowed service times for each customer. Thus, the pricing subproblem is equivalent to the shortest path problem with time windows and capac-

ity constraints. Another dynamic programming approach is proposed by Nagih and Soumis [22] for the shortest path problem with resource constraints. The aim is to find the least costly path from the source node to the sink node such that the resource constraints are satisfied. However, as the number of resources increases, the complexity (thus the time required to solve the problem) increases quickly. A heuristic is proposed to reduce the size of the resource space by using Lagrangian and surrogate relaxation methods. Desrosiers *et al.* [15] also present dynamic programming solution methods for time and resource constrained shortest path problems. While trying to minimize the total traveling cost, the solution should satisfy the allowed time intervals specified for each node. Each path on the network is identified by two labels: cost and time. Two algorithms are proposed for this labeling scheme: The first one is based on label correcting. Each node is treated once and all paths of a treated node are carried to its successors through the arcs that connect these nodes. The second algorithm applies label setting and labels are treated instead of nodes. In this algorithm, the path that has the minimum time is selected and carried to the successor nodes. The advantage of this algorithm is that it works even if the cost parameters on the arcs or cycles in the network are negative. The number of labels defined for each problem is $n+1$ where n is the number of resources and one additional label is kept for the calculating the cost of the path. Domination rules are applied at each node in order to prune some unnecessary paths. Desaulniers *et al.* [11] adopt a branch-and-price-and-cut method for solving the VRPTW where the column generation subproblem is an *elementary shortest path problem with resource constraints (ESPPRC)*. They develop a tabu search heuristic for the pricing subproblem that generates negative reduced cost columns rapidly. In addition to this heuristic method, they introduce a generalization of the k-path inequalities and emphasize that these generalized inequalities can theoretically be stronger than the traditional ones. In their study, they also propose a new subproblem type called the *partially elementary shortest path problem with resource constraints (PESPPRC)* for which elementarity requirements are imposed only on a subset of the nodes. They show that the PESPPRC is easier to solve than the ESPPRC and it yields lower bounds that are comparable in quality with the ones provided by the ESPPRC. With their method which combines all of these ideas, they report solving five of the previously unsolved 100-customer benchmark instances of Solomon.

Label correcting algorithms are frequently employed in the shortest path problem literature. The *bi-criterion shortest path problem*, which is a subproblem for many

transportation and scheduling problems, is solved by Skriver and Andersen [25] utilizing a label correcting algorithm. Bi-criterion shortest path problem, unlike the classical shortest path problem which is easier to solve, considers two different objectives at the same time: minimizing the total cost and minimizing the total travel time. The algorithm proposed by Skriver and Andersen is an improvement of Brumbaugh-Smith *et al.*'s algorithm [8] which utilizes several labels to solve the bi-criterion shortest path problem. *Multi-criteria shortest path problem*, which is even harder than the bi-criterion shortest path problem, is studied by Guerriero and Musmanno [18]. On random networks, they test several label correcting methods based on either node-selection or label-selection and show that label-selection algorithms perform better than node-selection methods for networks with high-density.

For the airline crew pairing problem, using the reduced cost as the pricing criterion is common practice. Anbil *et al.* [2] use the reduced cost criterion for their shortest path column generation scheme. They apply depth-first search on the duty tree in order to find negative reduced cost pairings. An alternate pricing rule based on a score obtained by dividing the pairing cost with the sum of the dual values of the flights in the pairing is introduced by Bixby *et al.* [6]. This rule is called the *lambda pricing rule* and *lambda* is calculated as follows for each partial pairing during column generation:

$$\lambda_p = c_p / \sum_{i \in p} y_i, \tag{2.1}$$

where c_p is the cost of pairing p and y_i is the dual value for the coverage constraint corresponding to flight leg i in pairing p. Notice that the dual values are summed over all flight legs covered by pairing p. λ value for each pairing with negative reduced cost pairing is calculated. Then, k columns with the k lowest λ_p are sent to the restricted master problem instead of the columns with the most negative reduced costs. In this study, it is shown that applying this selection rule reduces the number of iterations. Makri and Klabjan [21] also use the score criterion proposed by Bixby *et al.*.

A pairing p has negative reduced cost if and only if its corresponding λ_p is less than one. Two types of pruning rules are utilized to avoid total enumeration of pairings and these rules are called *exact* and *approximate rules*. The approximate rules fathom partial pairings that will likely result in a pairing with a score greater than or equal to one, i.e. $\lambda_p \geq 1$. However, optimality is not guaranteed by the approximate rules

since these rules might fathom some pairings that would yield negative reduced cost. Therefore, if the approximate rules fail to find a pairing with negative reduced cost, the column generation algorithm resorts to the exact rules for pricing.

CHAPTER 3

PRICING BY LOCAL SEARCH IN COLUMN GENERATION FOR THE AIRLINE CREW PAIRING PROBLEM

3.1 Problem Statement

In the crew pairing problem, the number of all feasible pairings grows exponentially as the number of flight legs in the schedule increases. Therefore, the generation of all pairings becomes an intractable task and it is out of the scope of this study. Instead of enumerating all possible pairings for a given network, we adopt a column generation approach for the set covering formulation of the crew pairing problem. Due to the very large number of variables (pairings), column generation is a suitable approach for this formulation and traditionally, the pricing subproblem for the column generation in the crew pairing problem corresponds to the multi-label shortest path problem (MLSP). However, the large scale of the search space is an important issue again at the MLSP phase, since solving MLSP becomes very time-consuming for moderate to big flight networks, and hence, the pricing subproblem turns out to be the bottleneck of the whole column generation algorithm. In this thesis, we propose a time-efficient pricing procedure involving a local search (LS) over the duty period pool (See Section 3.3.3) and a heuristic MLSP procedure over the partial duty network (See Section 3.3.2) as to avoid executing the MLSP algorithm at each iteration of the column generation. We make use of the duty period pool we accumulate during the initial feasible pairing generation phase and the MLSP phase in order to create negative reduced cost pairings in a reasonable time as well as to reduce the number of times the MLSP algorithm is called during the entire column generation algorithm. In order to represent the flight schedule, either a flight network or a duty network is used [26]. Throughout this study, we make use of both networks. We work on the flight network to denote the flights and the possible connections and to execute the exact MLSP algorithm. A small flight network example is illustrated in Figure 3.1. We utilize the (partial) duty network in order to find negative reduced cost pairings faster than in the MLSP solved over the

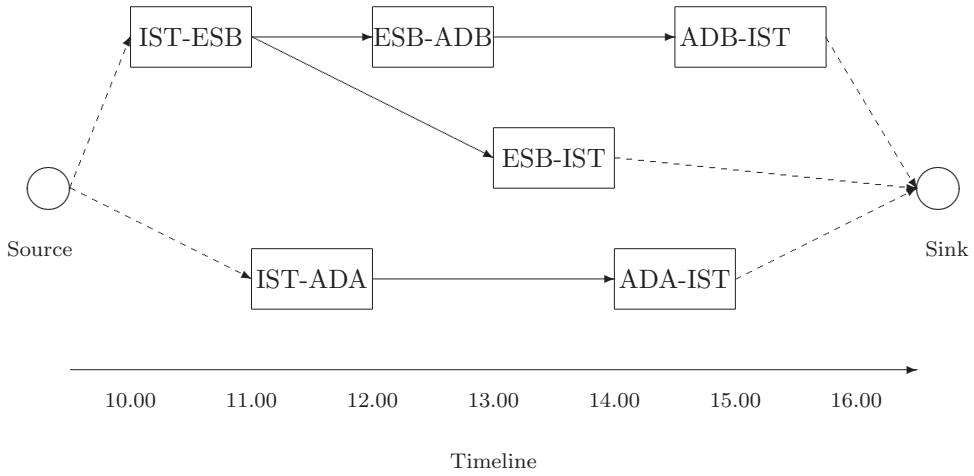

Figure 3.1: A small flight network example.

flight network. Details of the heuristic MLSP method on the duty network are given in Section 3.3.2 and Figure 3.2 exhibits a duty network example.

It should be noted that the flight network structure adopted in this study is different than the traditional flight network structure. Common practice in the crew pairing literature is to represent each flight leg by two nodes (*departure node* and *arrival node*) and to connect them by a *flight arc*. Connections are denoted by *connection arcs*. However, in our flight network construction, each flight leg in the schedule is represented by one node in the network and these nodes are connected to each other by connection arcs following the two basic sit connection rules:

1. The arrival city of the first *Flight* is the same with the departure city of the second *Flight*.

2. The duration of the sit connection cannot exceed the maximum sit time, nor it can be below the minimum sit time. The maximum sit time is 4 hours and the

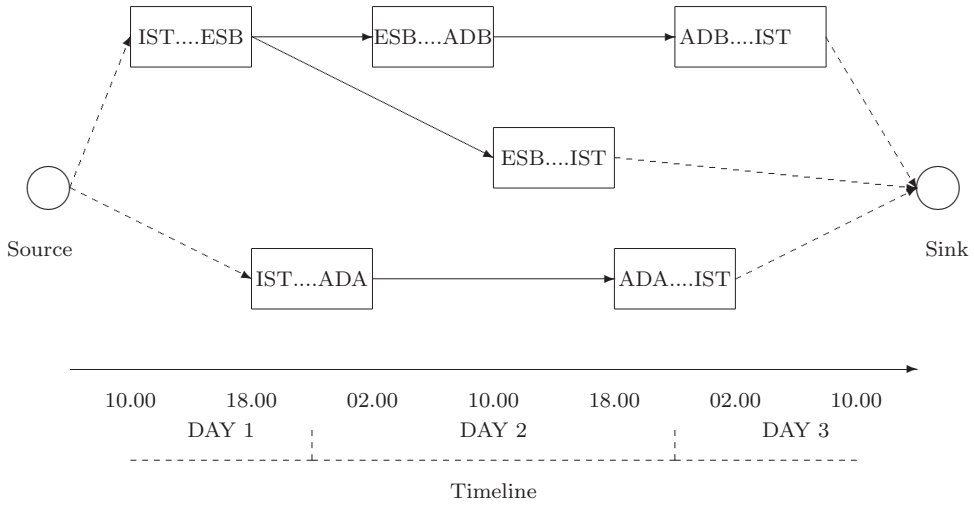

Figure 3.2: A small duty network example.

minimum sit time is 20 minutes in our case.

As the MLSP algorithm is executed on the flight network, several other feasibility rules should be tracked. Table 3.1 shows the minimum rest time needed between two duty periods and the maximum elapsed time allowed for a duty period is shown in Table 3.2.

Duration of the Previous Duty Period	**Minimum Rest Time Between Two Duty Periods**
Less than 4 Hours	8 hours
4 to 11 Hours	10 hours
11 to 12 Hours	12 hours
12 to 14 Hours	14 hours
18 hours or more	20 hours

Table 3.1: Minimum rest time between two duty periods.

	Number of Flight Legs		
Starting Time of the Duty Period	**1-3 Flights**	**4-5 Flights**	**6 or more Flights**
05.00-14.00	14 h	13 h	12 h
14.01-17.00	13 h	12 h	11 h
17.01-04.59	12 h	11 h	10 h

Table 3.2: Maximum elapsed time for a duty period.

Similar to our flight network structure, our adopted duty network structure is also different than that of the classical crew pairing literature. This time, each node represents one duty period (basically a sequence of flight legs connected to each other subject to some feasibility rules) and arcs are used to connect these duty periods. It is important to note that less number of feasibility rules should be tracked on the duty network since duty period related requirements are already met during duty construction. This allows faster traversal on the network, therefore provides potential to price out negative reduced cost pairings more rapidly.

In addition to the nodes that represent flights and duty periods, there are also a dummy source and a dummy sink node in both networks. These two nodes are needed in order to construct feasible paths (pairings) in the network starting from the source and ending at the sink. All arcs emanating from the dummy source correspond to a departure from the crew base and all incoming arcs to the dummy sink represent an arrival to the crew base. Note that the crew base is *IST* for the networks illustrated in Figure 3.1 and in Figure 3.2.

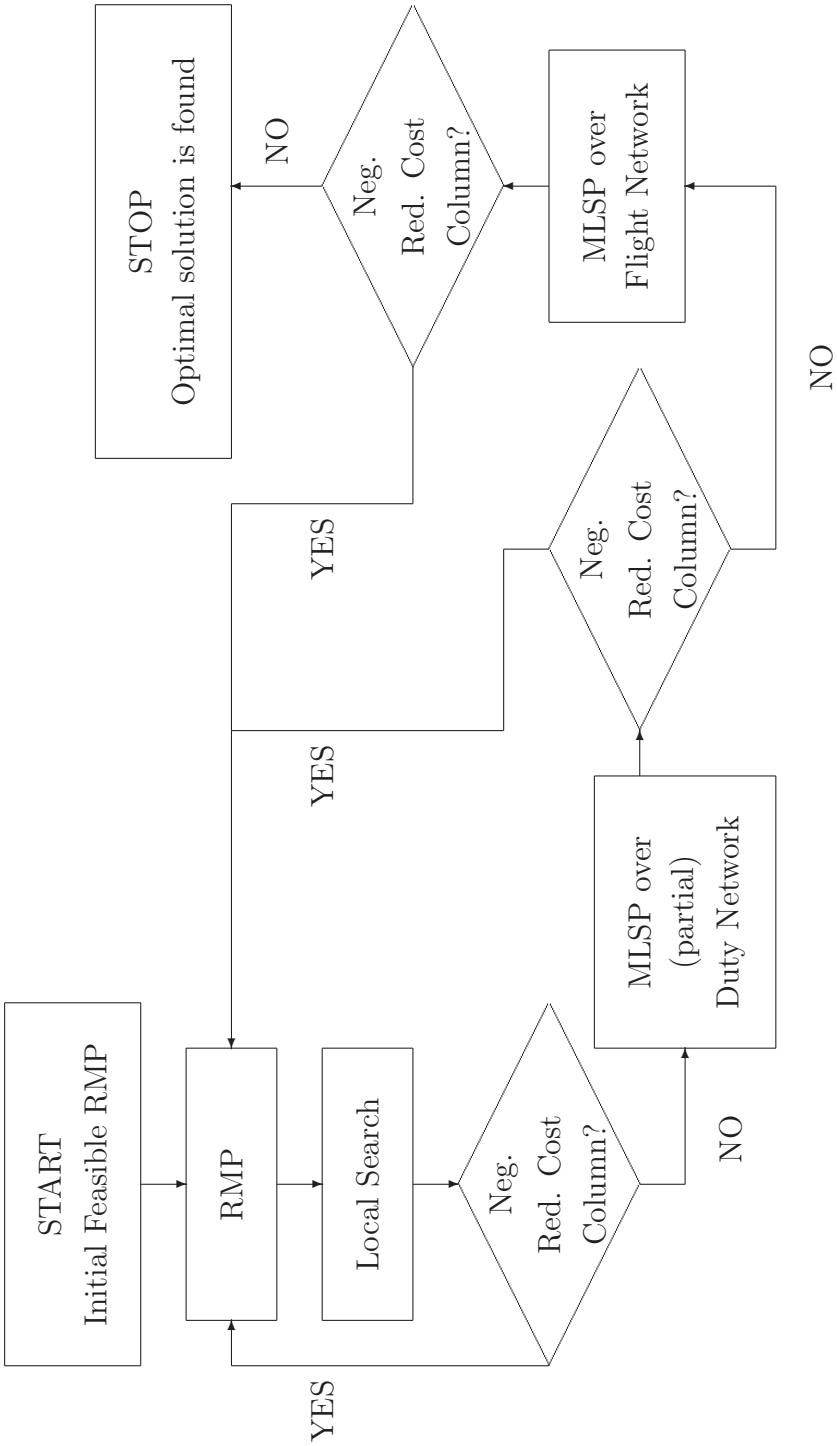

Figure 3.3: Flow chart of the proposed algorithm.

For our proposed local search based pricing heuristic (See Section 3.3.3), generating a diverse set of duty periods is crucial. Throughout this thesis, we refer to this set as the *duty pool*. The majority of the duty periods in the duty pool is generated during the initial feasible pairing pool generation phase (See Section 3.2). Each time the MLSP algorithm adds a new pairing to the restricted master problem, the duty periods that this pairing consists of are added to the duty pool so that the duty pool is updated for the subsequent iterations. As shown in Table 3.2, duty periods are also subject to some feasibility rules and the feasibility of the duty periods added to the duty pool is ensured during the update. A flow chart of our proposed method is illustrated in Figure 3.3. Details of each step are explained in the subsequent sections.

3.2 Initial Feasible Pairing Pool Generation

To start the main flow of the algorithm, a feasible set of pairings that cover all flight legs is essential. With this set of pairings, the RMP is solved for the first time, and the dual values for all flight legs are initialized. Within the column generation algorithm, the pricing phase introduces new pairings (columns) to the RMP until the optimal solution is found.

An initial feasible pairing pool for the RMP can be generated using an artificial pairing method. In this method, an artificial pairing, which consists of only one flight leg, is created for each flight leg in the schedule (so that all flights are covered and the set covering model is feasible) and very high costs are assigned to these artificial pairings. These high-cost pairings are eventually removed from the optimal solution of the RMP as the pricing subproblem finds favorable feasible pairings that actually exist in the flight network. In this study, the artificial pairing approach is not adopted for the following reasons:

- The artificial pairing technique generally maintains poor initial objective function values, since the costs of the artificially generated pairings are very high. A significant amount of time is spent trying to remove these pairings from the RMP and this consequently increases the number of MLSP iterations needed to reach the optimal solution.

- The basis of our local search algorithm is the duty pool mentioned in Section 3.1. With the artificial pairing technique, an initial diverse duty pool cannot be generated, since most of the involved pairings do not actually exist. Therefore,

duty periods can only be accumulated during the MLSP invocations over the flight network, and the diverse duty pool necessary for the local search cannot be created.

Due to the issues mentioned above, a different initial feasible pairing pool generation technique is proposed. Essentially, our proposed method involves solving a series of multi-label shortest path problems on the flight network until all flights in the schedule are covered. Here, the shortest path is determined according to the *visiting cost* defined for each node in the network. These visiting costs are initially zero for all flights. After each MLSP iteration, the pairing with the minimum total visiting cost is added to the initial pairing pool, its duty periods are added to the duty pool and the visiting costs of all flight legs (nodes) covered by the pairing (path) are increased by a predefined large number. As the iterations progress, the visiting costs of the covered flight legs increase rapidly and the paths with more uncovered flight legs are favored. Feasibility tracking for duty periods and pairings is performed by the same procedure as that of the MLSP algorithm (See Section 3.3.1). Basically, at each iteration, the MLSP is solved following the same label updating procedures with Algorithm 3.2. The only difference is that the shortest path here is determined according to the visiting cost, whereas in Algorithm 3.2, the path with the minimum reduced cost is the shortest path.

Algorithm 3.1: Initial feasible pairing pool generation algorithm.

1: **Initialize:**
$visitCost_i = 0 \ \forall i \in \mathcal{F}$
2: **while** there are uncovered flights **do**
3: Solve MLSP on the flight network
4: Add pairing j with the smallest total visiting cost to the initial pairing pool
5: Add all duty periods covered by pairing j to the duty pool (if these are not duplicates of already existing duties)
6: Increase visiting costs of all flight legs covered by pairing j
7: **end while**

3.3 The Pricing Subproblem

The pricing subproblem essentially corresponds to finding feasible pairings which will improve the objective function of the RMP. In our problem, we start the column generation algorithm with a subset of all possible feasible pairings that is generated by using Algorithm 3.1. In the main loop, the RMP and the pricing subproblem are solved alternatingly until the pricing procedure fails to find a negative reduced cost

pairing (i.e. optimal solution is reached). Each time the RMP is solved, the dual values corresponding to the flight legs change, and the pricing subproblem tries to find the favorable pairing according to these updated values. The dual interpretation of the pricing step can be explained using the dual of the LP relaxation of the primal set covering model:

$$\text{maximize} \sum_{i \in \mathcal{F}} u_i \tag{3.1}$$

$$\text{subject to} \sum_{i \in \mathcal{F}} a_{ij} u_i \leq c_j, \qquad j \in \mathcal{P}, \tag{3.2}$$

$$u_i \geq 0, \qquad i \in \mathcal{F}, \tag{3.3}$$

where u_i corresponds to the dual variable of flight (constraint) i in the primal problem. u_i values are obtained after each reoptimization of the RMP and are used to calculate the reduced cost of the pairing that will be added to the RMP in the next iteration. The reduced cost $\bar{c_j}$ of any pairing j is given by:

$$\bar{c_j} = c_j - \sum_{i \in \mathcal{F}} a_{ij} u_i. \tag{3.4}$$

Unless we have a degenerate LP, any negative reduced cost pairing found after pricing improves the primal objective function value once added to the RMP.

Solving the MLSP is the core component of our pricing procedure. It provides negative reduced cost pairings to the RMP, and ensures that the feasibility rules are satisfied by tracking several labels on the (partial) duty periods and the (partial) pairings. Domination rules ensure that dominated partial paths are not carried until the sink node and the number of alternative paths is reduced which makes it faster to reach to the sink node.

In this study, multi-label shortest path algorithm is applied on two different networks: the partial duty network and the flight network. MLSP over the duty network is a heuristic approach based on finding shortest paths on the partial duty network (which is constructed using the duty period pool) in order to price out the negative reduced cost pairings. This approach is faster than MLSP over the flight network, which is an exact approach based on total implicit enumeration of the feasible pairings. Since the search is carried out on a partial network, MLSP over the duty network may fail to find a negative reduced cost pairing even if such a pairing exists. In that case, exact

MLSP over the flight network is required to ensure that if a negative reduced cost pairing cannot be found, the optimal solution to the LP relaxation is found.

The other key procedure to obtain negative reduced cost pairings is the local search algorithm applied over the duty pool. The proposed local search algorithm is a heuristic method for pricing out the promising pairings to be added to the RMP. The local search algorithm provides a simple and fast way of providing negative reduced cost columns to the RMP and as long as it succeeds, MLSP is not employed for pricing. However, since the local search algorithm is a heuristic way of looking for negative reduced cost columns, it does not guarantee the optimality of the RMP. Thus, MLSP and the local search algorithm are used alternatingly. The former is required to prove the optimality of the subproblem while the latter helps speeding up the iterations and solving the whole problem in less number of iterations.

3.3.1 Pricing by Multi-Label Shortest Path Algorithm on the Flight Network

As its name implies, the multi-label shortest path problem is different from the shortest path problem with a number of labels kept throughout the nodes in order to record the state of the path. In the crew pairing problem, these labels are important for tracking feasibility conditions related to pairings and duty periods. In general, a label is used for each metric mentioned in the feasibility rules of the problem. In addition to these, the dual values and the costs are also traced. On each node in the network, there is a node label consisting of several path labels. In our case, as the flight network is traversed, these path labels are updated according to the following atomic labels (i.e. attributes) attached to each one of them:

- Total elapsed time
- Total cost
- Sum of the dual values of the flight legs covered
- Number of flight legs covered
- Number of completed duty periods

Figure 3.4 shows how each node label is structured. Let us assume there are m different atomic labels needed to track the information about a path. For the sample node label illustrated in Figure 3.4, there are n different paths (thus n different path

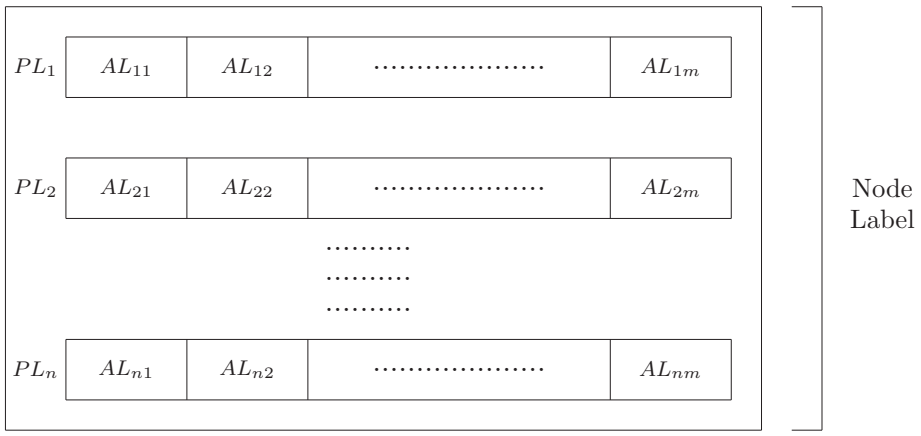

Figure 3.4: Label structure for the MLSP.

labels) from the source node to this node. Recall that each node in the network is represented with a single node label consisting of several path labels and the atomic labels are what the path labels are made of. Defining the set of different atomic labels on a path label as A and the set of path labels on the node label as L, atomic label $i \in A$ on path label $j \in L$ is denoted by AL_{ji} while the path label j is denoted by PL_j.

Each duty period on any path is also checked for feasibility during MLSP and duty period information is stored. Implementation details for network and label structures are given in Section 3.4.

As Figure 3.4 illustrates, each node label may consist of several path labels. In other words, there may be more than one path emanating from the source node to any intermediate node. However, not all of these partial paths are required to be extended until the sink node. A partial path is pruned (i.e. no longer considered) either if it violates the feasibility conditions or if it is dominated by any other path. Domination criterion, which is specific to our case, is explained by Definition 1. To clarify this definition, let us denote the state of a path i on some node j with $(AL^j_{iPL_i}, RC^j_i)$ where PL_i is the set of all labels kept on path i. $(AL^j_{iPL_i})$ is listed as $(AL^j_{i1}, AL^j_{i2}, ..., AL^j_{i|PL_i|})$ and RC^j_i is the reduced cost of path i on node j. Then,

Definition 1 *We say that, among two paths represented by two path labels PL_1 and PL_2 and emanating from the source node to node j with states $(AL^j_{1PL_1}, RC^j_1)$ and $(AL^j_{2PL_2}, RC^j_2)$, respectively, PL_1* **dominates** *PL_2 if and only if $RC^j_1 \leq RC^j_2$ and $AL^j_{1l} \leq AL^j_{2l}$, $\forall l \in PL$. In this case, the paths represented by PL_1 and PL_2 are called* **nondominated** *and* **dominated** *paths, respectively.*

Note that, in our flight network construction phase, the topological order of the nodes is ensured. That is, connections are always established from left to right along the time line (See Figure 3.1). In Algorithm 3.2, the nodes are processed following their topological order. Nondominated paths on each node are carried to the successor nodes using the necessary information retrieved from these two nodes as well as from the arc connecting them. To illustrate the way labels are updated on an example, let us denote the label representing the total elapsed time on any path label as ${L_i}^{et}$ and ${L_j}^{et}$ for nodes i and j, respectively, the duration of the flight leg represented by node j as dur_j, and the duration of the connection (i.e. sit time) between these two nodes as $SitTime_{ij}$. Then, the label update is done as follows:

$$ {L_j}^{et} = {L_i}^{et} + dur_j + SitTime_{ij}. \tag{3.5} $$

Algorithm 3.2 explains how the nodes are processed during one MLSP iteration. The output of Algorithm 3.2 is the pairing with the minimum reduced cost which corresponds to the path from the source node to the sink node with the minimum reduced cost. Note that our proposed algorithm (See Figure 3.3) resorts to MLSP over the flight network if and only if it fails to find a negative reduced cost pairing after searching the duty network using MLSP.

In Algorithm 3.2, n denotes the number of nodes in the flight network. v_i represents the i^{th} node in the flight network (according to the topological order), and k_{ij} is the j^{th} adjacent node of node v_i while the number of nodes adjacent to node v_i is denoted by $\mathcal{K}_i$.

Algorithm 3.2: Multi-label shortest path algorithm for the flight network.

1: **Initialize:**
Set null node labels $\forall i \in \mathcal{F}$
2: **for** $i = 1$ to n **do**
3: Apply domination rules to path labels on v_i and prune dominated paths
4: **for** $j = 1$ to $\mathcal{K}_i$ **do**
5: Update all path labels on k_{ij} through the arc from v_i to k_{ij}
6: Check feasibility rules for the updated path labels and prune infeasible paths
7: **end for**
8: **end for**
9: Sort all paths at the sink node according to their reduced costs
10: **Output:** The path with the minimum reduced cost

3.3.2 Pricing by Multi-Label Shortest Path Algorithm on the Duty Network

In the airline crew pairing problem, a duty network consists of duty periods represented by nodes and rest connections represented by arcs. The network structure is similar to the flight network however, the node structure is different. As it is mentioned in Section 3.1, while one node of the flight network denotes a single flight leg, one node of the duty network stands for a number of flight legs concatenated under duty period feasibility rules (i.e. a feasible duty period). Similarly, the arcs of the duty network correspond to overnight rest connections whereas one flight network arc might either denote a short sit connection or a longer rest connection. One common characteristics for the two networks is that, in both networks, one path from the source node to the sink node constitutes a feasible pairing. Therefore, solving the multi-label shortest path problem over the duty network is an alternative way for pricing out negative reduced cost pairings for the restricted master problem. However, it should be noted that, the duty network utilized in this study is a partial duty network. The reason for working on a partial network is that the total enumeration of all possible duty periods is an exhaustive task and it is almost equivalent to generating all possible pairings. Since we look for a time-efficient solution method for the airline crew pairing problem, constructing the whole duty network is not viable. Instead, we work on a partial duty network constructed from the duty pool generated during the initial feasible pairing pool generation phase and updated each time a new pairing is added to the RMP by the MLSP solved over the flight network. In the subsequent sections, our partial duty network will simply be referred to as the *duty network*.

Solving the multi-label shortest path problem on the duty network has a potential to efficiently generate negative reduced pairings for the following reasons:

- The duty periods in the duty pool are known to be feasible and ensuring the feasibility of the duty periods reduces the number of labels kept during the MLSP. Therefore, during the multi-label shortest path algorithm, only the pairing related feasibility rules should be tracked. As a result, having less number of labels makes the multi-label shortest path problem easier to solve on the duty network.

- As the pairings are generated, the duty pool is updated by accumulating all distinct duty periods. The duty pool is used in order to construct the duty network, where the duty periods are denoted by the nodes of the network. Compared to

the flight network, the duty network, even the partial one, might include a larger number of nodes. At first, this might seem to be a shortcoming of the network structure, since it would increase the size of the network. However, in the meantime, the duty network approach decreases the depth of the search tree. In other words, it is easier to reach to the sink node from the source node, since there is a smaller number of nodes that need to be traversed. Likewise, the length of a pairing is reduced in terms of the nodes it consists of. This is very important for the multi-label shortest path problem since it tries to find the shortest path from the source to the sink as quickly as possible. To illustrate this concept on an example, let us consider a pairing consisting of eight flight legs. This corresponds to a 10-node (including the source and the sink) path on the flight network. Let this pairing be separated into three feasible duty periods with overnight rests between them. Then, the corresponding pairing would be a 5-node (again including the source and the sink) path on the duty network. Rather than traversing 10 nodes to reach to the sink node, a 5-node traversal would be enough to constitute the same pairing. The reduction in the depth of the search tree saves time and effort, and allows a faster execution of the MLSP on the duty network than that on the flight network.

The motivation behind constructing the duty network is the opportunity to work with a smaller and compact version of the flight network so that the search space is reduced and exhaustive enumeration of all pairings is avoided. It should be noted that the multi-label shortest path algorithm explained in Section 3.3.1 is still essentially valid. Algorithm 3.3 exhibits the slightly modified version of Algorithm 3.2 for the duty network.

In Algorithm 3.3, n denotes the number of nodes in the duty network. v_i represents the i^{th} node in the duty network (according to the topological order), and k_{ij} is the j^{th} adjacent node of node v_i while the number of nodes adjacent to node v_i is denoted by $\mathcal{K}_i$. Here, $\mathcal{D}$ is the set of all duty periods in the partial duty network.

Notice that label updating and node treating procedures are the same in both algorithms. The duty network version outperforms the flight network version due to the reduced number of feasibility rules and tracked labels as well as the decreased depth of the search space, reducing the time needed for a MLSP execution. However, since the duty network is partial and the whole space is not explored, the multi-label shortest path algorithm over the duty network is not sufficient to announce the optimality of

Algorithm 3.3: Multi-label shortest path algorithm for the duty network.

Initialize:
Set null node labels $\forall i \in \mathcal{D}$
for $i = 1$ to n **do**
  Apply domination rules to path labels on v_i and prune dominated paths
  for $j = 1$ to $\mathcal{K}_i$ **do**
    Update all path labels on k_{ij} through the arc from v_i to k_{ij}
    Check feasibility rules for the updated path labels and prune infeasible paths
  end for
end for
Sort all paths at the sink node according to their reduced costs
Output: The path with the minimum reduced cost

the overall problem.

3.3.3 Pricing by A Local Search Algorithm

Since the multi-label shortest path problem is *NP*-hard, some of its instances can be very hard to solve depending on several factors such as the number of flight legs in the schedule, the general network structure, the structure of the feasibility rules and the number of labels associated with these rules. Thus, it is common practice to try to avoid solving the MLSP subproblem to optimality at each column generation iteration and to apply heuristic methods for the pricing subproblem. On the other hand, heuristic methods are not sufficient to prove the optimality of the current restricted master problem and solving the subproblem to optimality is necessary at some point during the iterations. Therefore, we resort to MLSP when our heuristic local search pricing algorithm fails to find a negative reduced cost pairing. This way, we aim to reduce the number of times MLSP is executed and the total computation time for the column generation by rapidly generating negative reduced pairings for the restricted master problem in a simple way.

Our proposed local search algorithm is based on deletion and insertion operators applied to a promising set of pairings utilizing the duty pool generated during the initial feasible pairing pool generation phase and updated as new pairings are added to the problem. Here, the promising set of pairings are defined as the set of pairings associated with the basic variables of the current restricted master problem solution. These are good candidates to turn into negative reduced cost pairings by simple operations since they already have zero reduced costs. At each iteration of the column generation, after solving the restricted master problem, the set of pairings with zero reduced costs is processed using Algorithm 3.4. For Algorithm 3.4, *dutyPool* denotes the set of duty

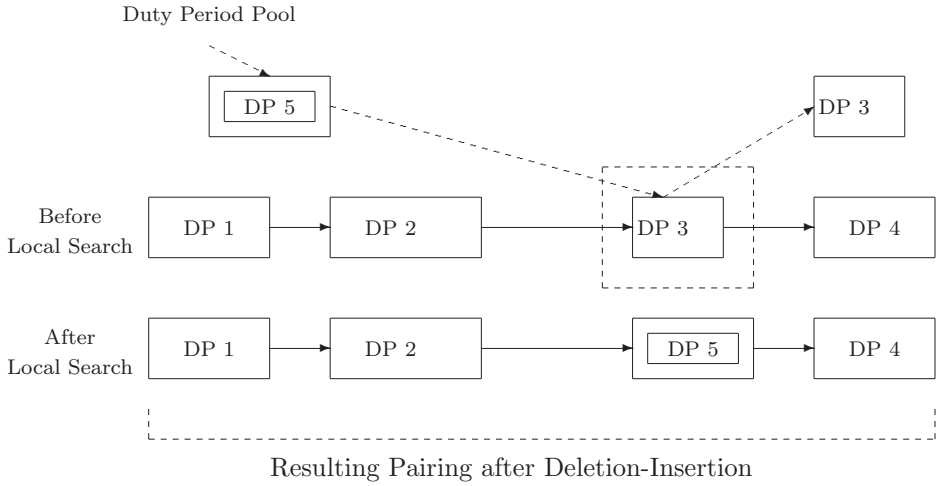

Figure 3.5: Local search method.

periods in the duty pool. $ZRCP$ represents the set of zero reduced cost pairings and $NRCP$ represents the set of negative reduced cost pairings. The i^{th} pairing in $ZRCP$ is illustrated by $ZRCP_i$ and $ZRCP_{ik}$ is the k^{th} duty period in $ZRCP_i$. n is the number of zero reduced cost pairings in $ZRCP$, while m denotes the number of duty periods in $dutyPool$.

Algorithm 3.4: Local search algorithm.

Input: $ZRCP$, $dutyPool$
Initialize:
$NRCP = \varnothing$
for $i = 1$ to n **do**
  for $j = 1$ to m **do**
    for $k = 1$ to number of duty periods in $ZRCP_i$ **do**
      Try:
      Delete duty period $ZRCP_{ik}$ from $ZRCP_i$
      Insert $dutyPool_j$ to the position $ZRCP_{ik}$
      if insertion is feasible **then**
        Calculate reduced cost for the new pairing
        if the reduced cost for the new pairing is negative **then**
          Insert new pairing to $NRCP$
        end if
      end if
    end for
  end for
end for
Output: $NRCP$

3.4 Implementation Details

Besides the hybrid pricing algorithm we propose for the airline crew pairing problem, we utilize a generic programming scheme to code and test our methods. The class design and the implementation with Boost Graph and Date-Time Libraries [7, 24] provide a neat and efficient way to deal with complex network-related operations.

3.4.1 Class Design and Structures

There are two basic types of structures used in our class design: flight network related structures and label structures. The basic building block for our flight network design is the *FlightLeg*. Each *FlightLeg* is represented by its:

- Departure City
- Arrival City
- Departure Time
- Arrival Time
- Duration
- Index (i.e. its place in the given timetable)

One or more *FlightLeg*s come together in order to form *SequenceOfFlightlegs*. Each *SequenceOfFlightlegs* carries certain elements such as:

- Total Cost
- Total Elapsed Time
- Total Flight Time
- Covered Flight Legs
- Node Label

A *SequenceOfFlightlegs* is either a *DutyPeriod* or a *Flight*. As their names imply, a *DutyPeriod* contains more than one *FlightLeg* while a *Flight* has a single *FlightLeg*. Each flight in the schedule represented by a *Flight* node in the network and these nodes are connected to each other by *Connection* arcs. Each *Connection* is denoted by its:

- Total Cost
- Duration
- Tail Node (i.e. the *Flight* that the *Connection* starts from)
- Head Node (i.e. the *Flight* that the *Connection* ends at)

Notice that each *SequenceOfFlightlegs* has a node label member. This corresponds to the other component of our class design: label structures. A *NodeLabel* is attached to all nodes in the flight network and as shown in Figure 3.4, a *NodeLabel* is made of several *PathLabel*s consisting of the following *AtomicLabel*s:

- Total elapsed time
- Total cost
- Sum of the dual values of the flight legs covered
- Number of flight legs covered
- Number of completed duty periods

The information enclosed in the *PathLabel* structure is essential in the pricing procedure since feasibility tracking and domination decisions for all paths are made according to this information. According to the feasibility requirements, the number of *AtomicLabel*s encapsulated by a *PathLabel* can be increased or decreased and new types of path information can be introduced thanks to the flexible and generic structure of our label design.

3.4.2 Generic Programming with Boost Libraries

Using the structures explained in Section 3.4.1, all of the algorithms were implemented using the Visual C++ programming language. In order to construct and manipulate the flight network, we make use of Boost Graph Library (BGL), a generic open source C++ library and an efficient tool for graph abstraction in a flexible environment. Due to the generic nature of BGL, graph structures are defined as templates and this allows the user to work with any data structure she wants. In our work, *SequenceOfFlightlegs* is given as the vertex type and *Connection* is given as the edge type of the graph. Network construction is carried out following the built-in BGL procedures.

We also utilize Boost Date-Time Library in order to easily work with date and time structures. Since the nature of the crew pairing problem is strictly based on temporal constraints and the network structure is dependent on date and time related rules, the functionality of the date-time library is of great importance. Boost Date-Time Library provides the user a robust set of operators and calculation capabilities. Therefore, date/time comparisons, adding/subtracting time durations/intervals and retrieval of times and dates from clocks are carried out simply and efficiently.

We refer to [7, 24] for other useful Boost C++ libraries and further information on Boost Graph and Date-Time Libraries. We present class diagrams and structures in Appendix A.

CHAPTER 4

COMPUTATIONAL RESULTS

In this chapter, we present the results of the numerical study on a set of real-life data acquired from a local airline company. We solve four problem instances consisting of **200**, **236**, **281** and **300** flight legs. In the subsequent sections, these data sets will be referred to as **DS1**, **DS2**, **DS3** and **DS4**, respectively. As it is explained throughout this thesis, the number of all possible pairings is very large for each instance. Therefore, the column generation approach, of which the details are explained in Chapter 3, is adopted to solve these instances. The algorithm is implemented using the Visual C++ programming language, and Boost C++ Libraries [7, 24] are used to manipulate the network structures. The restricted master problem of the LP relaxation for the set covering model is optimized by ILOG CPLEX 12.1 [19].

In order to present numerical results that attest to the efficiency of our approach, we benchmark our algorithm against a pure MLSP pricing scheme. As it is exhibited in Figure 3.3, our adopted algorithm breaks down the pricing subproblem into three steps. First, a local search mechanism is applied on a set of pairings. In cases when the local search cannot construct a negative reduced cost pairing (in other words, none of the pairings it constructs has negative reduced cost), the multi-label shortest path problem is solved over the duty network. If this second heuristic approach also fails, the multi-label shortest path problem is solved over the flight network. Since this last alternative is an exact enumeration method, we are assured that the optimal solution is reached if the MLSP algorithm over the flight network cannot find a negative reduced cost pairing anymore. Therefore, our termination criterion is satisfied when the exact method (MLSP over the flight network) fails to add a negative reduced cost pairing to the RMP. While our hybrid algorithm uses three different pricing methods alternatingly, the pure MLSP scheme, as its name implies, solves the pricing subproblem by executing the multi-label shortest path algorithm over the flight network at each subproblem iteration. For illustrative purposes, our proposed algorithm will be referred to as the

hybrid method and the pure MLSP pricing scheme will be denoted by the *pure method* in the subsequent sections. We will make use of the following abbreviations for the sake of simplicity:

- **LS:** local search mechanism
- **DNMLSP:** multi-label shortest path problem over the duty network
- **FNMLSP:** multi-label shortest path problem over the flight network
- **CG:** column generation
- **DP:** duty period pool
- **HYBRID:** our hybrid pricing scheme
- **PURE:** pure FNMLSP pricing scheme

Benchmarking our algorithm against a pure FNMLSP pricing scheme is crucial since we want to measure the effect of incorporating LS and DNMLSP into the pricing mechanism. FNMLSP is an *NP*-hard problem and our main goal should be to avoid solving this problem at each and every subproblem iteration if possible. This way, we expect the following to occur:

- the number of column generation iterations that FNMLSP is executed would be reduced,
- the computation time per iteration would be reduced due to the heuristic methods (i.e. LS and DNMLSP),
- the total computation time needed to solve the whole instance would be reduced due to the reduction in computation time per iteration.

Therefore, we present the results based on a comparison between PURE and HYBRID approaches. One should note that the initial feasible pairing pool generation mechanism (See Section 3.2) is the same for both approaches. A series of multi-label shortest path problems is solved for four instances until all flight legs are covered and an initial feasible pairing pool is generated. CPU times needed to generate an initial feasible solution are presented in Table 4.1.

It is mentioned in Section 3.2 that the quality of the initial feasible solution (i.e. the objective function value of the initial RMP) is important in order to reach the optimal

	DS1	DS2	DS3	DS4
CPU sec.	70.95	302.94	439.60	424.48

Table 4.1: CPU times needed to generate an initial feasible solution.

solution more rapidly. In Table 4.2, the initial and optimal objective function values of the RMP are exhibited for four instances.

Throughout this thesis, we deal with the LP relaxation for the set covering model of the airline crew pairing problem. That is, we relax the integrality constraints (1.3) in the set covering formulation, and allow the variables to take non-integer values. Our proposed column generation solution method is based on this relaxed model, and the optimal objective function values we report in Table 4.2 are the optimal objective function values for the LP relaxation of the RMP, thus, they are lower bounds on the integer programming optimal objective function values. In Table 4.2, we also present the IP optimal objective function values (with respect to the restricted set of pairings we have at the end of the CG) for DS1, DS2, DS3, and DS4. The pairings (columns), which are generated during the column generation iterations are given as input to the IP set covering model and the model is optimized using ILOG CPLEX 12.1 [19]. It should be noted that the optimality gaps between the LP solutions found by our proposed method and the set covering IP solutions are significantly small.

	DS1	DS2	DS3	DS4
Initial OFV	11102.2	14203.5	17817.1	18464.2
Optimal OFV (LP)	7530	8635	11030	11210
Optimal OFV (IP)	7530	8670	11190	11220

Table 4.2: Initial and (LP and IP) optimal objective function values.

Figures 4.1, 4.2, 4.3 and 4.4 give the changes in the durations of subproblem iterations for DS1, DS2, DS3 and DS4, respectively. In these charts, red data points correspond to the iterations where only LS is used for pricing. Blue data points represent the iterations where LS is unsuccessful but DNMLSP is successful at finding a negative reduced cost pairing. The iterations at which both heuristic methods (LS and DNMLSP) fail and hence, FNMLSP is executed are denoted by the yellow data points. Considering the distribution of these data points over the timeline, the following observations can be made:

- Let t_{LS}, t_{DNMLSP} and t_{FNMLSP} denote the computation time needed for a subproblem iteration when, respectively, LS, DNMLSP and FNMLSP are executed.

Then, the relationship between t_{LS}, t_{DNMLSP} and t_{FNMLSP} values can be defined as $t_{LS} < t_{DNMLSP} < t_{FNMLSP}$. This result is expected since both LS and DNMLSP are heuristic methods and they aim to find a quick solution to the pricing subproblem by reducing the search space but they both are short of proving optimality. FNMLSP is an exact method which is based on exploring the whole search space, thus, it is the most time consuming method. Comparing LS with DNMLSP, LS outperforms DNMLSP in terms of computation time since DNMLSP has a more sophisticated search mechanism than LS. While LS looks for negative reduced cost pairings only among the basic variables of the RMP (i.e. pairings with zero reduced cost), DNMLSP considers all pairing alternatives that can be generated using the duty network generated up until then.

- LS mechanism is successful at the early phases of the CG and stalls after some point. This is caused by the characteristics of the duty period pool. The duty period pool is populated with a rich and diverse number of duty periods at the initial feasible pairing pool generation phase (See Section 3.2) and after this phase, the duty period pool diversity cannot be increased much. As a result, LS mechanism operates on similar pairings with almost the same duty periods at each iteration. Therefore, the chance of creating a new negative reduced cost pairing is decreased.

- DNMLSP is generally successful at finding negative reduced cost pairings. However, it also stalls at some point during the iterations (at later phases of the CG) and the pricing subproblem resorts to FNMLSP frequently after this point. As it is mentioned above, although DNMLSP works on a broader search space than LS, it is still a heuristic method and the duty network is partial. Thus, FNMLSP execution is crucial at some point during the iterations, since it maintains the optimality of the subproblem.

- Tracing the yellow data points, it is observed that FNMLSP executions last significantly longer at the early and late FNMLSP iterations. At the early iterations, there exist plenty of negative reduced cost columns (pairings) that are candidates for entering the basis. This causes the MLSP algorithm to spend more time to find the most negative one among these candidates. The number of candidate pairings decreases in the subsequent iterations, and the computation times per iteration are reduced as expected. However, after some point during the itera-

tions, at later phases, we again observe an increase in the computation time per CG iteration. This time, the increase is caused by the large number of pairings whose reduced costs are costs are negative and very close to zero. Due to numerical precision, these numbers are not considered as zero and the paths that they are associated with cannot be dominated. For this reason, the number of non-dominated paths increases and it takes more time to reach the sink node.

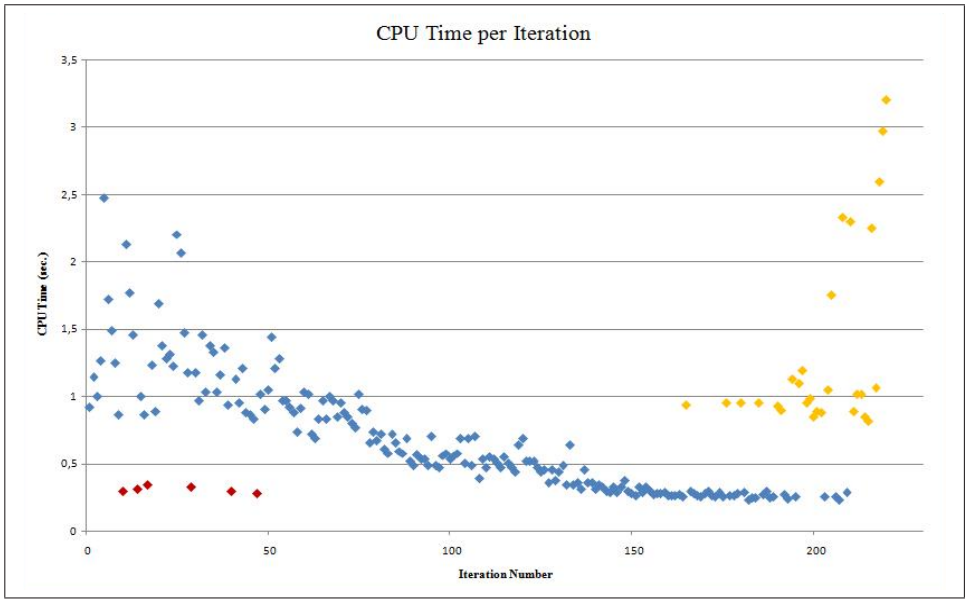

Figure 4.1: CPU times (sec.) per iteration for DS1.

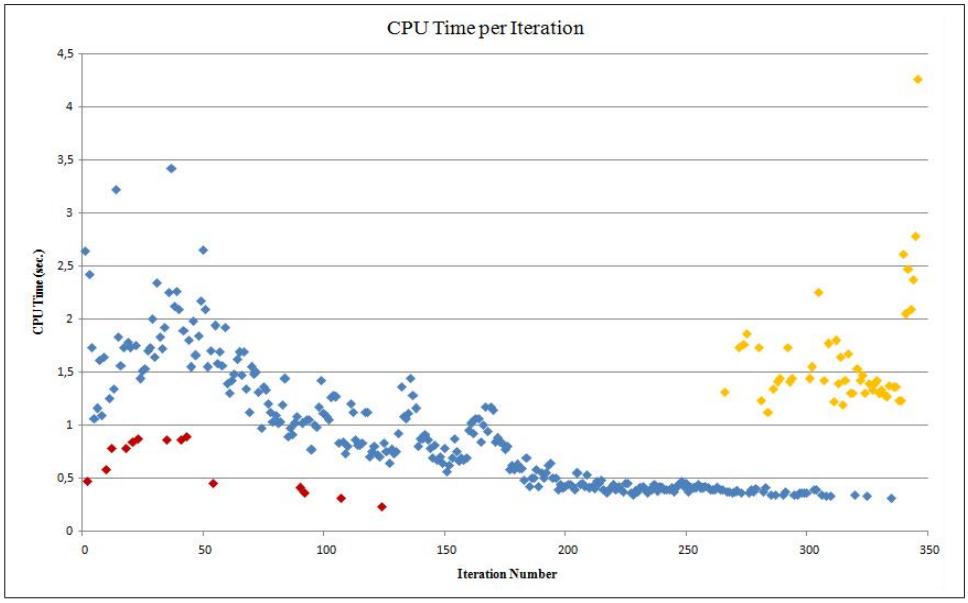

Figure 4.2: CPU times (sec.) per iteration for DS2.

In Tables 4.3, 4.4, 4.5 and 4.6 pure and hybrid methods are compared in terms of the total computation time needed to solve the problem, the number of FNMLSP executions and the number of CG iterations. It is important to note that the reported

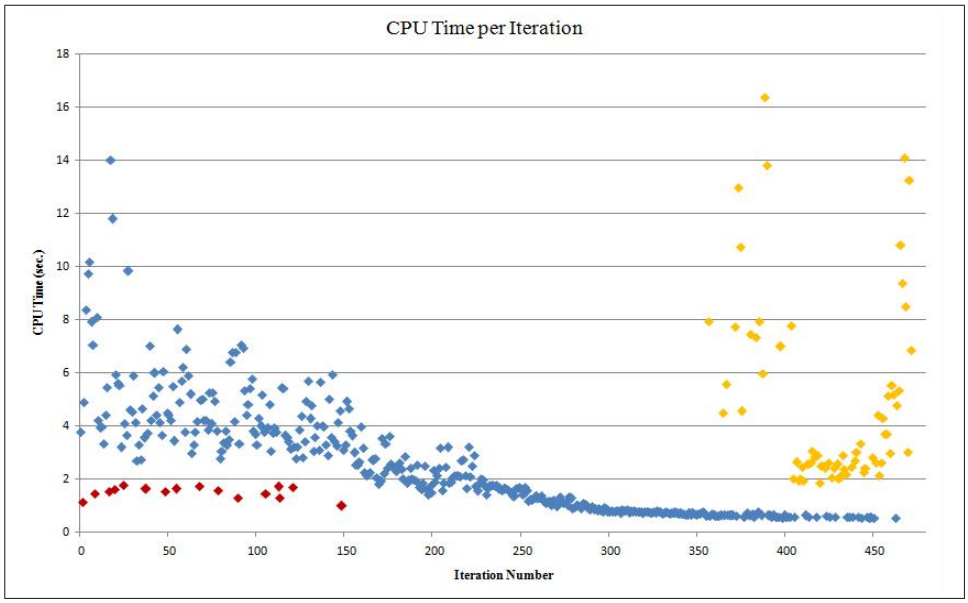

Figure 4.3: CPU times (sec.) per iteration for DS3.

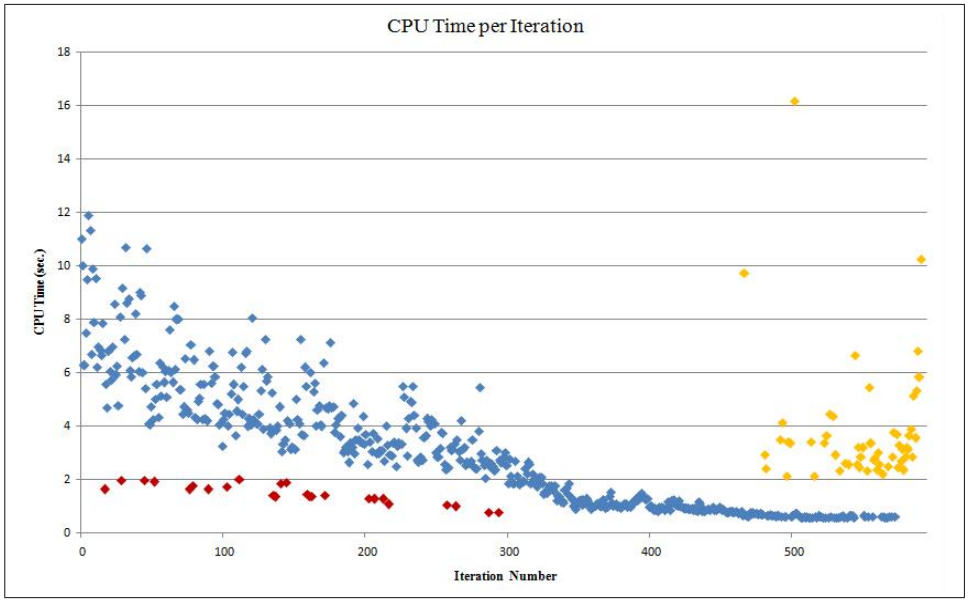

Figure 4.4: CPU times (sec.) per iteration for DS4.

total computation times exclude the time needed to generate an initial feasible solution (those are reported separately in Table 4.1). One can notice that the outcomes are as expected. By adopting the hybrid approach, the total CPU times are reduced. There is a significant reduction in the number of FNMLSP executions compared to the pure method. This exhibits the positive effect of LS and DNMLSP on the pricing mechanism. Another observation is that the total number of CG iterations might differ for two methods. In other words, the number of iterations needed to reach the optimal solution need not to be the same for each instance. This is again expected, since two methods price out different pairings for the restricted master problem. As different pairings enter to the basic solution, the objective function values might differ for some specific

CG iteration. For example, for DS2, at iteration 203, the objective function value is 9650 with the hybrid method, whereas it is 9192 with the pure method. However, at the last iteration of the CG, the objective function values for both methods should be equal, since the objective function value for the last CG iteration is the optimal objective function value.

	DS1	
	PURE	**HYBRID**
total CPU sec.	175.859	132.422
number of FNMLSP executions	213	28
number of CG iterations	213	220

Table 4.3: Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS1.

	DS2	
	PURE	**HYBRID**
total CPU sec.	442.188	366.89
number of FNMLSP executions	346	49
number of CG iterations	346	346

Table 4.4: Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS2.

In order to observe the effect of incorporating the local search into the pricing mechanism more clearly, we conduct a computational study in which we turn off the LS mechanism and run the pricing step by only utilizing FNMLSP and DNMLSP. Benchmarking the values in Table 4.7 and Tables 4.3, 4.4, 4.5 and 4.6, one may conclude that LS has a positive effect on the total computation time and the number of FNMLSP executions. These two values are between the two values acquired by PURE and HYBRID methods whereas the same positive effect is not observed on the total number of CG iterations. The total number of CG iterations is generally larger than those in both the PURE and the HYBRID methods.

We mention in Section 3.2 that a rich and diverse initial duty period pool is important for the efficiency of our approach. As it is exhibited in Algorithm 3.1, we populate

	DS3	
	PURE	**HYBRID**
total CPU sec.	1243.39	1115.31
number of FNMLSP executions	426	63
number of CG iterations	426	472

Table 4.5: Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS3.

	DS4	
	PURE	**HYBRID**
total CPU sec.	1587.34	1328.7
number of FNMLSP executions	558	55
number of CG iterations	558	591

Table 4.6: Comparison of HYBRID and PURE by total CPU times, number of FNMLSP executions and number of CG iterations for DS4.

	DS1	**DS2**	**DS3**	**DS4**
total CPU sec.	150.09	415.65	1210.32	1347.13
number of FNMLSP executions	28	59	77	57
number of CG iterations	224	364	481	585

Table 4.7: Some performance measures after turning off the LS mechanism.

the duty period pool by adding all of the duty periods covered by the pairing (if these are not duplicates of already existing duties) found at each iteration. We apply this step even if the new found pairing does not cover an uncovered flight leg but consists of duty periods that do not exist in the duty period pool. To illustrate this on an example, let us suppose that at iteration i of Algorithm 3.1, a generated pairing with a negative reduced cost consists of 3 duty periods. The first duty period consists of flight legs 2 and 5, the second duty period consists of flight legs 7, 9 and 11, and the third duty period consists of flight legs 15, 16 and 18. For illustrative purposes, let us denote this pairing i as $2-5/7-9-11/15-16-18$. Following the same notation, let the pairing generated at iteration $i+1$ be $2-7/11-16-18$. Notice that, pairing $i+1$ does not cover any new flight legs uncovered by pairing i. However, our initial feasible pairing pool generation mechanism adds all of the duties covered by these two pairings (i.e. $2-5$, $2-7$, $7-9-11$, $11-16-18$ and $15-16-18$). With this approach, we intend to generate a diverse duty pool. In order to measure the effect of the diversity of the initial duty period pool on our proposed algorithm, we modify our original initial feasible pairing pool generation technique as follows: We only consider new duty periods to be added to the pool if they come from a pairing that covers at least one flight leg that was previously uncovered. This way, some duty periods will not be able to enter the initial duty pool and the number of distinct duty periods will be reduced. This will reduce the diversity of the duty pool, and consequently, effect the performance of our pricing procedure. Table 4.8 provides a comparison between the modified and the original initial feasible pairing pool generation mechanisms. In the first row for each data set, the CG is started with a less number of duty periods in the duty pool, and the second row represents the results we find by our original technique,

explained by Algorithm 3.1. Comparing these two approaches, one can derive that reducing the number of duty periods in the duty pool (i.e. decreasing the diversity of the duty pool):

- negatively effects the performance of the LS,
- increases the number of CG iterations at which FNMLSP is executed,
- causes the first execution of the FNMLSP to shift to earlier iterations of the CG.

	initial # of duties in the DP	**# of iterations LS is successful**	**the iteration at which FNMLSP is first executed**	**# of total FNMLSP executions**
DS1	121	2	114	41
	249	6	165	28
DS2	149	3	169	76
	254	14	266	49
DS3	193	11	299	142
	325	16	357	63
DS4	211	9	310	123
	374	24	467	55

Table 4.8: Some performance measures for the proposed pricing method with the original initial feasible pairing pool technique and the modified version (with less number of duty periods).

The trend of the objective function values over time can be observed in Figures 4.5, 4.6, 4.7 and 4.8. Notice that the two methods follow different paths during the iterations, but at the last iteration, they coincide as expected. Pure FNMLSP scheme generally provides better quality pairings in terms of the amount of the improvement in the objective function value. On the other hand, the objective function value improvement is slower in the hybrid method. This difference is caused by the tradeoff between the solution quality and the time efficiency. With the hybrid method, we can provide negative reduced cost pairings more rapidly at each iteration at the expense of the magnitude of improvement in the objective function value. With the less time-efficient pure FNMLSP method, we explore the whole search space and enumerate all possible pairings at each iteration, which makes it more likely to price out good quality pairings.

From Figures 4.5, 4.6, 4.7 and 4.8, one can observe the sudden decrease in the objective function value acquired by the HYBRID method at some point during the iterations. This specific point corresponds to the iteration at which FNMLSP is first

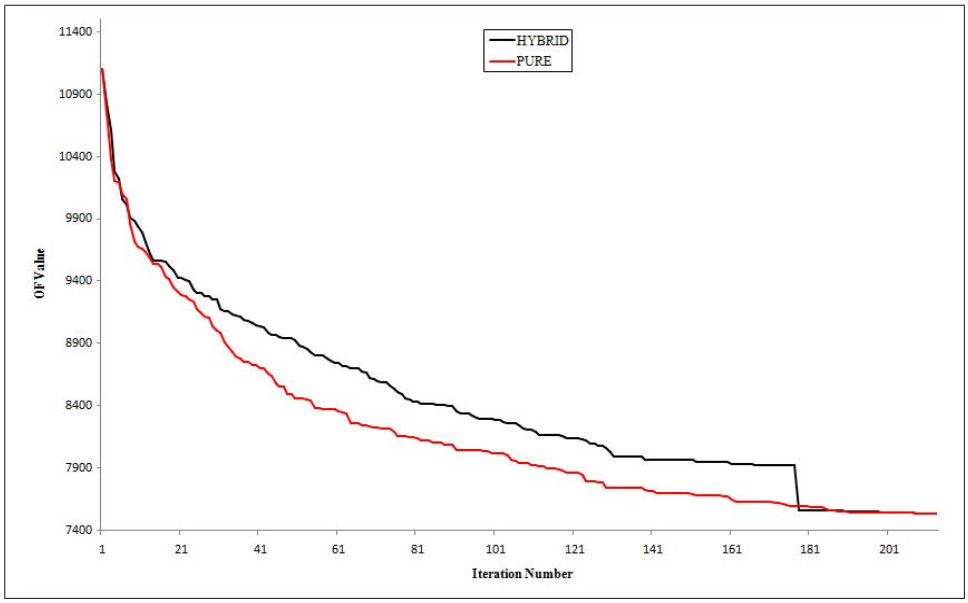

Figure 4.5: Comparison of HYBRID and PURE by changes in the objective function value for DS1.

executed. As Figures 4.1, 4.2, 4.3 and 4.4 also illustrate, after that very first execution of FNMLSP, LS and DNMLSP are not very successful at finding negative reduced cost pairings and FNMLSP is frequently executed for pricing. This causes unnecessary LS and DNMLSP executions and might negatively effect the total computation times. Additionally, after the sudden relatively large decrease in the objective function value, the improvement in objective function value is not significant. For benchmarking purposes, we present some results in Table 4.9 for which we turn off both LS and DNMLSP mechanisms and rely only on FNMLSP for pricing after the iteration at which FNMLSP is first executed. Notice that the performance of the algorithm is worse that of both the PURE and the HYBRID methods. This way, unnecessary LS and DNMLSP executions are avoided; however, the number of FNMLSP executions is increased causing longer total computation times.

	DS1	**DS2**	**DS3**	**DS4**
total CPU sec.	173.03	495.67	1360.89	1858.39
number of FNMLSP executions	48	58	115	73
number of CG iterations	213	324	472	541

Table 4.9: Some performance measures acquired by turning off the LS and DNMLSP mechanisms after the iteration at which FNMLSP is first executed.

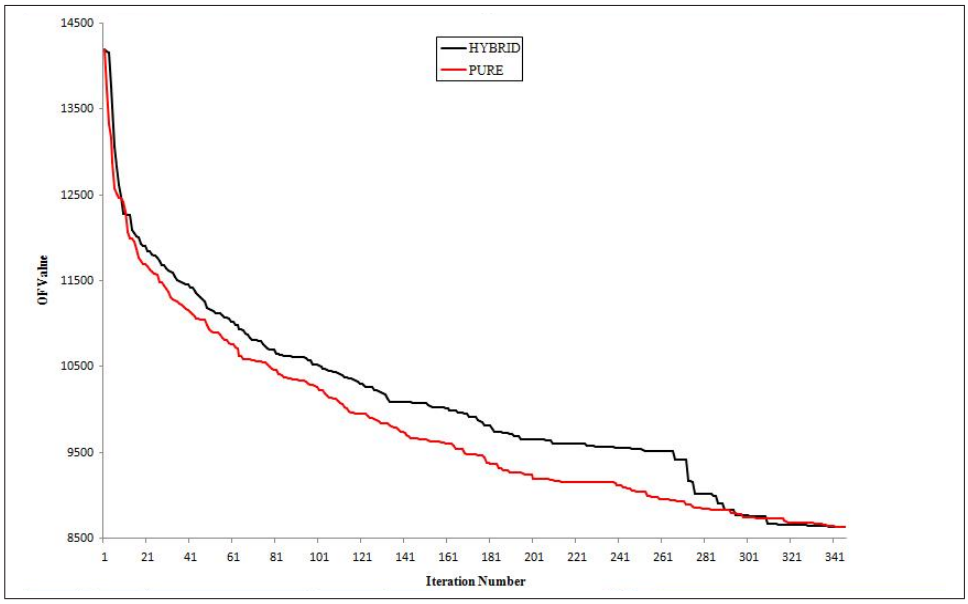

Figure 4.6: Comparison of HYBRID and PURE by changes in the objective function value for DS2.

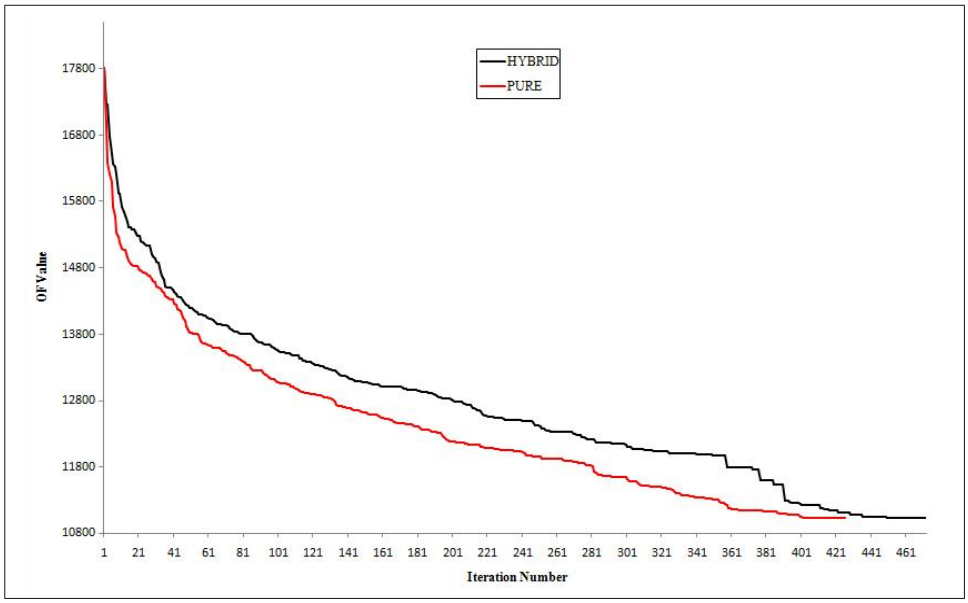

Figure 4.7: Comparison of HYBRID and PURE by changes in the objective function value for DS3.

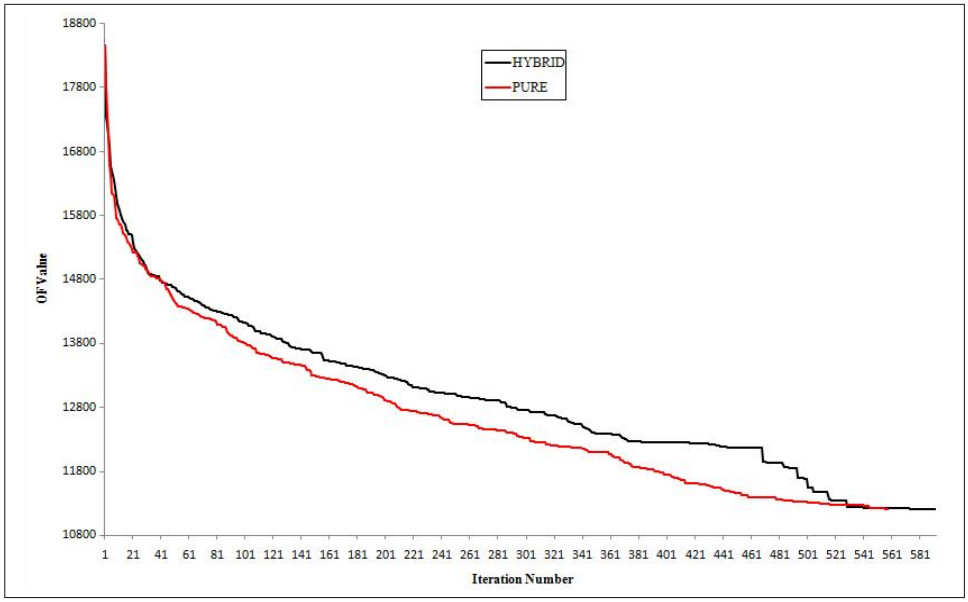

Figure 4.8: Comparison of HYBRID and PURE by changes in the objective function value for DS4.

CHAPTER 5

CONCLUSION

In this thesis, we solve the airline crew pairing problem by a column generation method. We particularly focus on the pricing subproblem of the column generation which is traditionally modeled as a multi-label shortest path problem solved over the flight network. This problem is *NP*-hard; thus, it is reasonable to try to avoid solving it at each subproblem iteration. To overcome the complexity of the pricing subproblem, we propose an alternate pricing scheme involving heuristic and exact methods. Instead of solving the multi-label shortest path over the flight network (which is an exact method), we first apply a local search mechanism to a set of zero reduced cost pairings using the duty period pool. In case of an unsuccessful local search, we solve the multi-label shortest path problem over the partial duty network. If this method also fails, we resort to the exact multi-label shortest path algorithm over the flight network ensuring the optimality of the subproblem.

We implement the proposed algorithm in a generic programming environment. We solve four real-life problem instances acquired from a local airline company and benchmark the performance of our method against a pure multi-label shortest path problem pricing scheme over the flight network. We observe reductions in the total computation times as well as the number of iterations an exact multi-label shortest path algorithm is executed. In addition, the feasible solutions of the crew pairing problem obtained from solving the set covering model over the set of pairings generated during the column generation are of high quality.

Bibliography

[1] Anbil, R., Gelman, E., Patty, B. and Tanga, R., Recent advances in crew-pairing optimization at American Airlines, Interfaces, 21, 62-74, 1991.

[2] Anbil, R., Forrest J.J. and Pulleyblank, W.R., Column generation and the crew pairing problem, Documenta Mathematica, Extra Volume ICM (3), 677- 686, 1998.

[3] Andersson, E., Housos, E., Kohl, N. and Wedelin, D., Crew pairing optimization, In G. Yu, editor, Operations Research in the Airline Industry, Kluwer Academic Publishers, 228-258, 1998.

[4] Barnhart, C., Jonhson, E.L., Nemhauser, G.L., Savelsbergh, M.W.P and Vance, P.H., Branch-and-price: Column generation for solving huge integer programs, Operations Research, 46(3), 316-329, 1998.

[5] Barnhart, C., Cohn, A.M., Johnson, E.L., Klabjan, D., Nemhauser, G.L. and Vance, P.H., Airline crew scheduling, In R. W. Hall, editor, Handbook of Transportation Science, Kluwer Scientific Publishers, 517-560, 2003.

[6] Bixby, R., Gregory, J.W., Lustig, I.J., Marsten, R. and Shanno, D., Very large scale linear programming: A case study in combining interior point and simplex methods, Operations Research, 40, 885-897, 1992.

[7] Boost C++ Libraries, 2010, http://www.boost.org/.

[8] Brumbaugh-Smith, J. and Shier, D., An empirical investigation of some bicriterion-shortest path algorithms, European Journal of Operational Research, 43, 216-224, 1989.

[9] Crainic, G. and Rousseau, J., The column generation principle and the airline crew scheduling problem, Informs, 25(2), 136-151, 1987.

[10] Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M. M. and Soumis, F., A unified framework for deterministic time constrained vehicle routing and crew scheduling problems, In T. Crainic and G. Laporte, editors, Fleet Management and Logistics, Kluwer Publishing Company, 57-93, 1998.

[11] Desaulniers, G., Lessard, F. and Hadjar, A., Tabu Search, Partial Elementarity, and Generalized k-Path Inequalities for the Vehicle Routing Problem with Time Windows, Transportation Science, 42(3), 387-404, 2008.

[12] Desrochers, M. and Soumis, F., A generalized permanent labeling algorithm for the shortest path problem with time windows, Infor, 26(3), 191-212, 1988.

[13] Desrochers, M. and Soumis, F., A column generation approach to the urban transit crew scheduling problem, Transportation Science, 23(1), 1989.

[14] Desrochers, M., Desrosiers, J. and Solomon, M. M., A new optimization algorithm for the vehicle routing problem with time windows, Operations Research, 40(2), 342-354, 1992.

[15] Desrosiers, J., Dumas, Y., Solomon, M. M. and Soumis, F., Time constrained routing and scheduling, In M. Ball, editor, Handbooks in Operations Research and Management Science, Elsevier, 35-140, 1995.

[16] Forrest, J.J., Mathematical programming with a library of optimization subroutines, presented at the ORSA/TIMS Joint National Meeting, New York, 1989.

[17] Gamache, M., Soumis, F., Morquis, G. and Desrosiers, J., A column generation approach for large-scale aircrew rostering problems, Operations Research, 47(2), 247-263, 1999.

[18] Guerriero, F. and Musmanno, R., Label correcting methods to solve multicriteria shortest path problems, Journal of Optimization Theory and Applications, 111(3), 589-613, 2001.

[19] ILOG software, http://www.ilog.com, July 2010.

[20] Kornilakis, H. and Stamatopoulos, P., Crew pairing optimization with genetic algorithms, Proceedings of the Second Hellenic Conference on AI: Methods and Applications of Artificial Intelligence, 109-120, April 11-12, 2002.

[21] Makri, A. and Klabjan, D., A new pricing scheme for airline crew scheduling, Informs Journal on Computing, 16(1), 56-67, 2004.

[22] Nagih, A. and Soumis, F., Nodal aggregation of resource constraints in a shortest path problem, European Journal of Operational Research, 172, 500- 514, 2006.

[23] Savelsbergh, M. and Sol, M., DRIVE: Dynamic routing of independent vehicles, Operations Research, 46(4), 474-490, 1998.

[24] Siek, J., Lee, L. and Lumsdaine, A., The Boost Graph Library User Guide and Reference Manual, Addison-Wesley, 2002.

[25] Skriver, A.J.V. and Andersen, K.A., A label correcting approach for solving bicriterion shortest-path problems, Computers and Operations Research, 27, 507-524, 2000.

[26] Vance, P.H., Atamtürk, A., Barnhart, C., Gelman, E., Johnson, E.L., Krishna, A., Mahidhara, D., Nemhauser, G.L. and Rebello, R., A heuristic branch-and-price approach for the airline crew pairing problem, 1997, http://citeseer.ist.psu.edu/vance97heuristic.html.

Appendix A

Class Diagrams and Structures

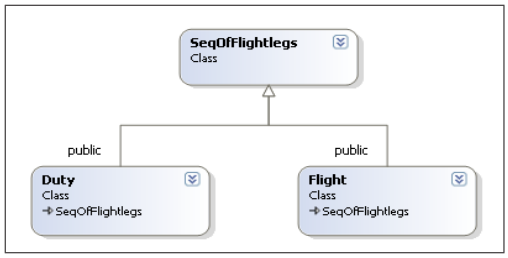

Figure A.1: *Duty* and *Flight* as *SeqOfFlightlegs*.

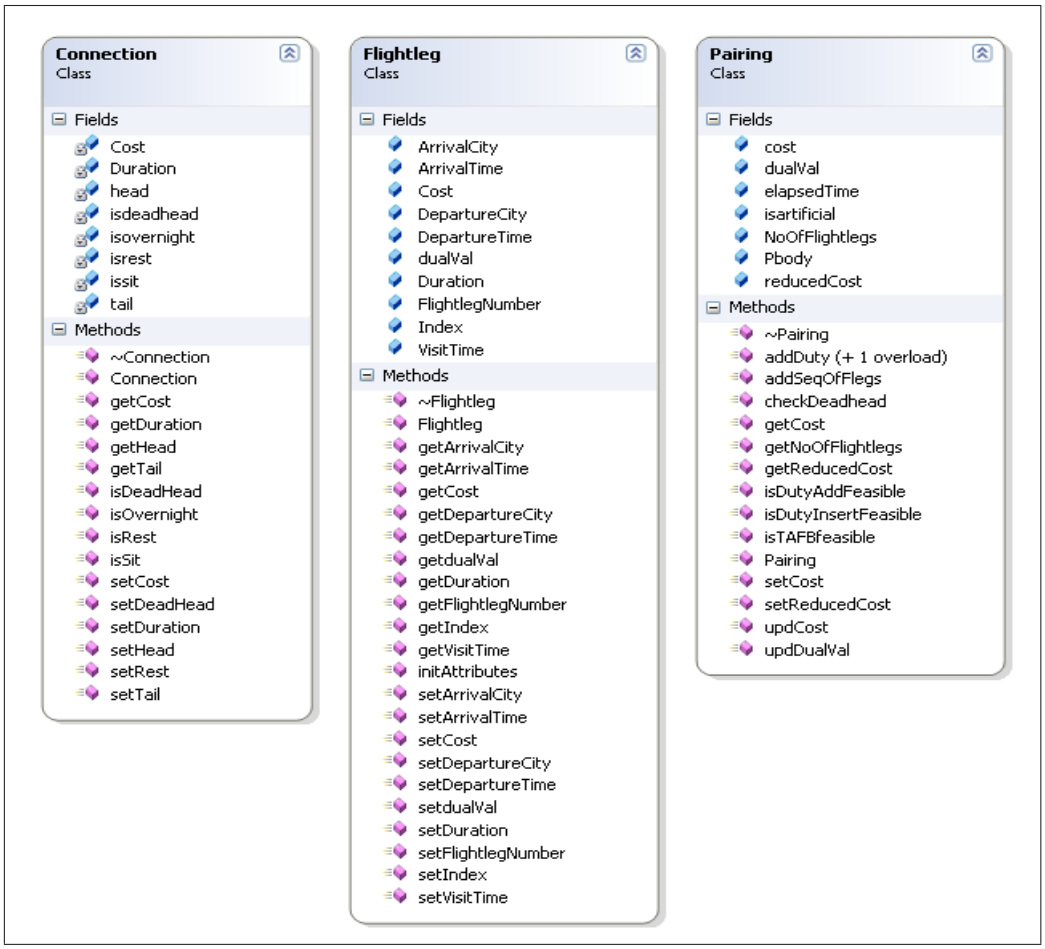

Figure A.2: *Connection*, *Flightleg* and *Pairing* classes.

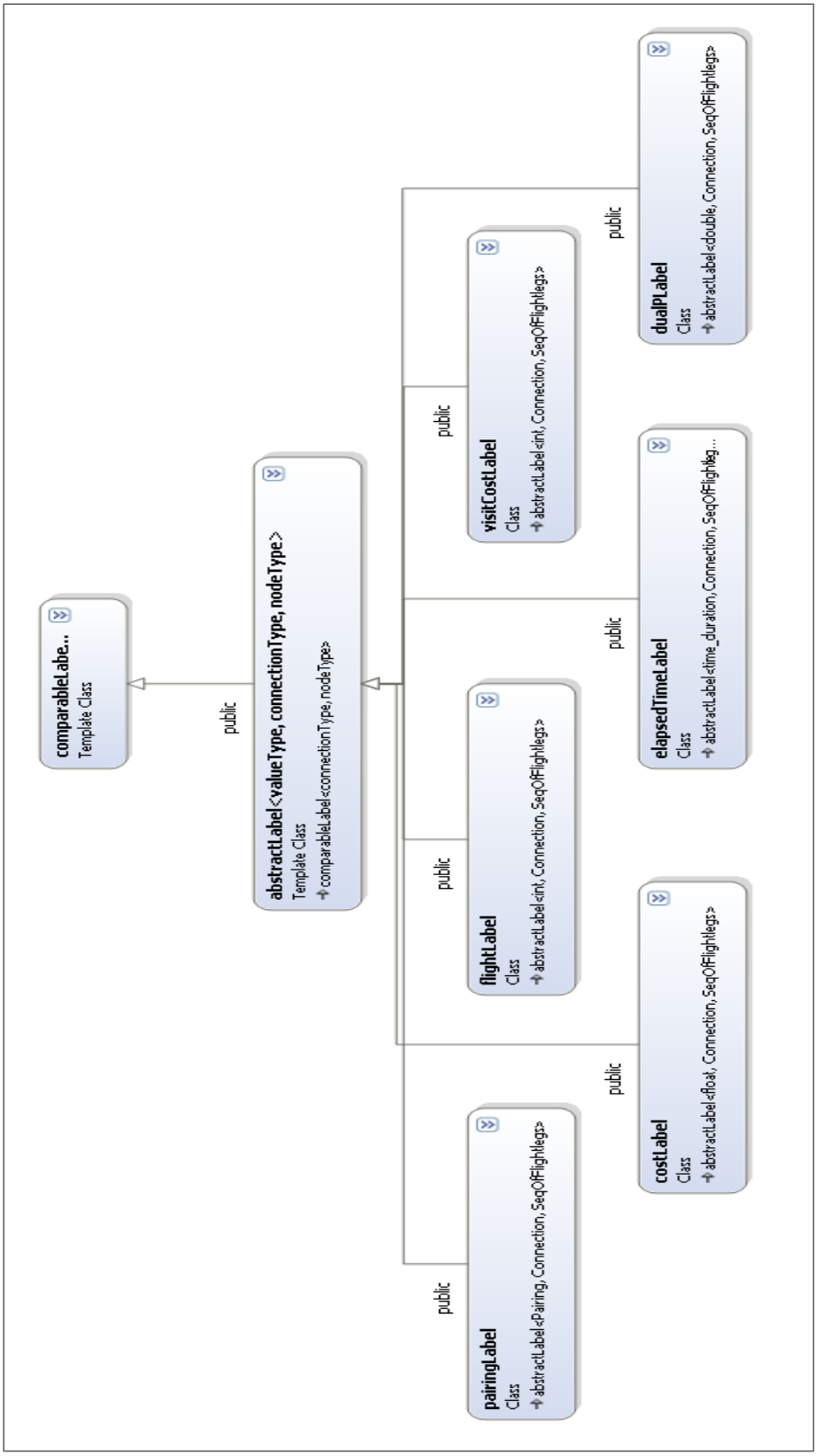

Figure A.3: Class diagram for the *Label* structure.

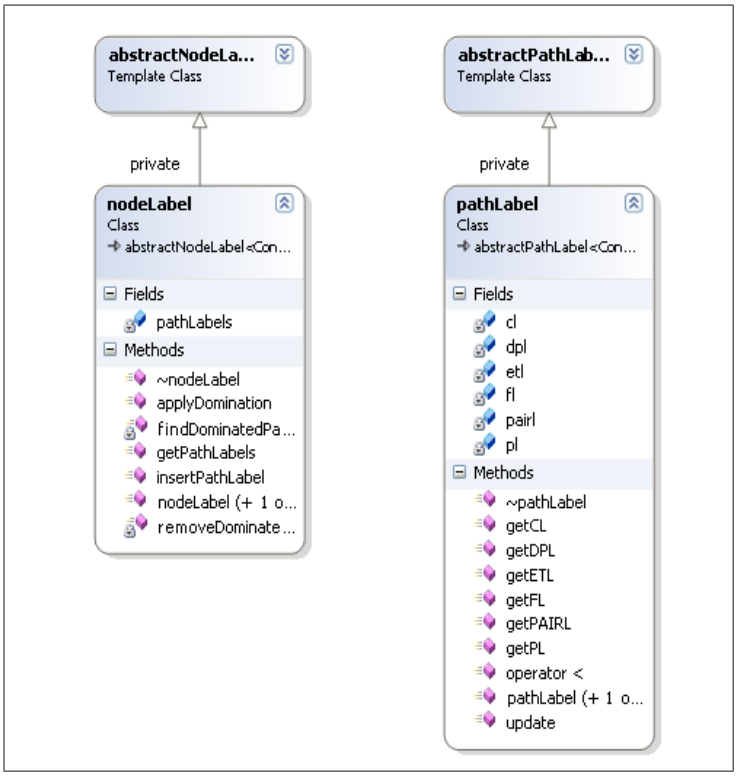

Figure A.4: *NodeLabel* and *PathLabel* classes.